

# Resumen de Teoría de las Telecomunicaciones

Guido Tagliavini Ponce

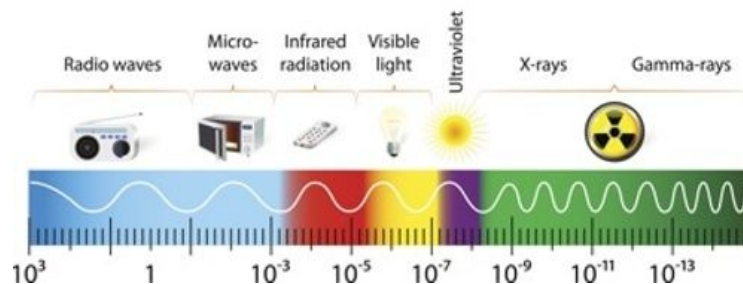
Última modificación: 11 de enero de 2017

Cursada del 2C-2016

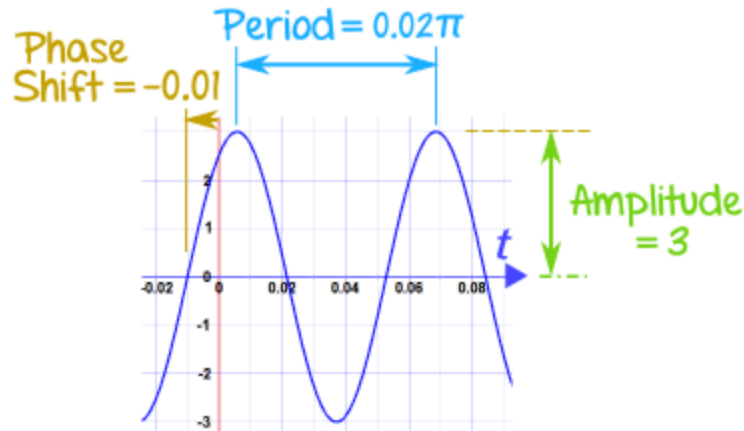
## Señales

- **Onda electromagnética:** campo eléctrico y magnético que se propagan juntos por un medio físico a una velocidad propia de éste.
  - Ejemplos: ondas de luz, de radio, microondas, rayos gamma, etc.

### THE ELECTROMAGNETIC SPECTRUM

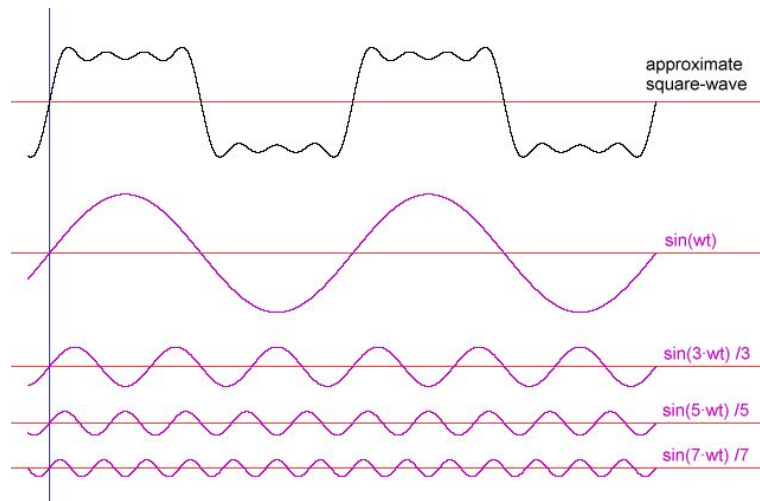


- La velocidad de propagación en el aire es  $c = 300000 \text{ km/s}$ , la velocidad de la luz.
- En otros medios se propaga más lento. Por ejemplo, en un cable de red típico la velocidad es  $0.69 * c$ .
- Las ondas electromagnéticas se describen matemáticamente.
  - Una **función periódica**  $f(t)$  es una función que cumple  $f(t) = f(t + T)$  para todo  $t$ . La mínima constante  $T$  que cumple lo anterior se llama **período fundamental**.
  - A una onda electromagnética la podemos describir como una función periódica  $s(t)$ .



- Por ejemplo:  $s(t) = A \sin(2\pi f \cdot t + c)$
- Características:
  - $s(t)$  = valor de la onda en el instante de tiempo  $t$
  - ciclo = tramo de la onda durante un período
  - $A$  = amplitud
  - $f$  = frecuencia = cantidad de ciclos por unidad de tiempo
    - Se mide en Hz (ciclos/seg).
  - $T$  = período = tiempo en el que se completa un ciclo
    - $T = 1/f$  (1 ciclo dividido la velocidad con la que se produce un ciclo).
  - $c$  = fase = desfase de la onda en el tiempo
  - $w$  = frecuencia angular =  $2f$  = frecuencia medida en radianes por segundo (2pi radianes = una vuelta a la circunferencia)
  - $v$  = velocidad de propagación de la onda
    - Esta magnitud no depende de  $s(t)$ . La función  $s(t)$  sólo indica cuál es el comportamiento de la onda, pero no dice nada sobre su velocidad de propagación en el medio. Podríamos tener una onda  $s(t)$  con una alta frecuencia, pero que se propague mega lento, como así también una  $s(t)$  con baja frecuencia, y que se propague mega rápido.
  - $\lambda$  = longitud de onda = distancia ocupada por un ciclo
    - Depende de la velocidad de propagación de la onda. A mayor velocidad de propagación
    - $\lambda = v \cdot T = v / f$
    - Si es  $v = c$  (por ejemplo, porque la propagación es en el vacío) se tiene  $\lambda = c / f$ .
- Idea: aproximar señales digitales usando señales electromagnéticas (analógicas).

- Ejemplo: sumando ciertas ondas seno, podemos aproximar una onda cuadrada, que es fácilmente interpretable como una señal digital.



- Serie de Fourier: toda función periódica puede expresarse como una serie infinita de la pinta

$$a_0 + \sum_{n=1}^{\infty} [a_n \cos(2 \pi (nf) * t) + b_n \sin(2 \pi (nf) * t)]$$

- El n-ésimo término es una suma de dos ondas, una seno y otra coseno, cada una con período nf.
- La frecuencia f (la de los senos y cosenos para n = 1) se llama **frecuencia fundamental**. El resultado de la suma es una onda que tiene frecuencia f.
- Los senos y cosenos del n-ésimo término se llaman **n-ésimos armónicos**.
- Como muestra la figura de arriba, la onda cuadrada puede formarse sumando los armónicos 1, 3, 5, 7, 9, ...
- Para aproximar una señal digital, sumamos senos y cosenos de la pinta de la serie de fourier, hasta tener una onda que se parezca a la señal digital que queremos transmitir.
  - Por “parecerse” queremos que decir que somos capaces de interpretar la señal digital como la señal analógica que queremos transmitir. Por ejemplo, a la señal digital de 1 y 0 alternados, podemos codificarla como la señal analógica que aproxima a la onda cuadrada. Los valores negativos de esta onda analógica son interpretados como los 0, y valores positivos como los 1.
  - En la práctica no podemos transmitir una suma infinita de senos y cosenos; sólo podemos sumar finitas. Esto implica que la aproximación no es perfecta. En general, cuantos más armónicos sumemos, mejor es la aproximación.

- En el extremo receptor de la onda analógica que transmitimos, podemos recuperar los coeficientes de cada uno los armónicos que transmitimos, gracias a más magia de Fourier. Esto permite, demultiplexar una señal que está transmitiendo dos ondas de distintas frecuencia. Por ejemplo, podríamos tener dos usuarios, uno que transmite una onda con frecuencia  $f_1$ , y otro una onda con frecuencia  $f_2$ , al mismo tiempo y sobre el mismo canal. Usando la magia de Fourier, en el extremo receptor podríamos descomponer la señal en las dos ondas de frecuencias  $f_1$  y  $f_2$ .
- **Ancho de banda**
  - El ancho de banda de una señal compuesta por ondas de frecuencias  $f_1 < \dots < f_n$ , es  $f_n - f_1$  (es decir, la máxima menos la mínima). En el caso de una suma de finitos armónicos, con período fundamental  $f$ , siendo el  $n$ -ésimo el armónico no nulo más grande, el ancho de banda es  $nf - f$ .
  - Se mide en Hz.
  - Frecuencia vs. ancho de banda
    - A mayor frecuencia de una onda, mayor información por unidad de tiempo puedo transmitir, pues los ciclos de la onda son los que nos permiten codificar información.
    - Por lo tanto, uno estaría tentado a usar una frecuencia altísima, para poder transmitir mucha información. Sin embargo, para transmitir información de forma confiable, vamos a tener que transmitir una onda tipo cuadrada. Esto nos obliga a refinar la onda de alta frecuencia, sumándole otras onditas con frecuencia más alta (otros armónicos). Esto implica que el ancho de banda crece.
    - Cuanto mayor es el ancho de banda, más tiempo toma producir y registrar la señal. Además no todos los medios de transmisión soportan cualquier ancho de banda. En general, vamos a tener una limitación ancho de banda.
    - Por lo tanto, a mayor frecuencia utilizada, mayor será el ancho de banda necesario para producir una señal confiable. Como el ancho de banda es limitado, la frecuencia a la que podemos transmitir también lo es.

## Nivel físico

- Al transmitir una señal por un medio, se pueden producir **perturbaciones** en la misma. La señal que llega al otro extremo no es exactamente la misma.

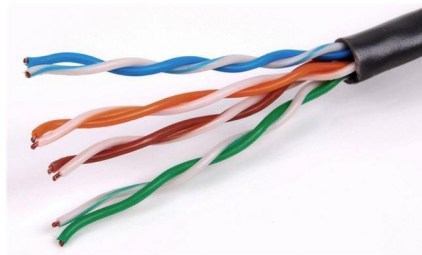
- Causas de las perturbaciones:
  - **Atenuación**
    - La intensidad de la señal disminuye con la distancia.
    - Depende del medio.
    - Las mayores frecuencias se ven más afectadas.
    - **Ecualización**: amplificar más las frecuencias más altas.
  - **Distorsión de retardo**
    - La velocidad de propagación en el medio varía con la frecuencia. La velocidad es mayor cerca de la frecuencia central (asumiendo que la señal está limitada en frecuencia).
    - Por lo tanto, las componentes de frecuencia centrales viajan más rápido al destino. Se producen desplazamientos de fase entre las distintas frecuencias.
    - Sólo ocurre en medios guiados.
  - **Ruido**
    - Señales adicionales insertadas entre el transmisor y el receptor.
    - **Térmico**
      - Debido a la agitación térmica de los electrones.
      - Aumenta linealmente con la temperatura absoluta.
      - Uniformemente distribuido sobre la frecuencia.
    - **Intermodulación**
      - Interferencia entre señales enviadas por el mismo medio.
      - Dos señales de frecuencias  $f_1$  y  $f_2$  pueden interferir con una señal de frecuencia  $n f_1 + m f_2$ .
    - **Diafonía**
      - Una señal de una línea interfiere con la señal de otra línea.
    - **Impulsivo**
      - Impulsos irregulares o picos.
      - De corta duración y gran amplitud.
- Transmisión de datos a través de un canal
  - La transmisión de datos es a través de una onda electromagnética que se propaga a través del canal. Por lo tanto, hay que codificar bits en una onda electromagnética.
  - **Velocidad de transmisión**. Cuántos bits por unidad de tiempo podemos codificar en una onda, por unidad de tiempo. Se mide en bps.
  - **Ancho de banda del canal**. Cuál es el ancho de banda del que disponemos. Está limitado por el transmisor y el medio.
  - **Tasa de errores**. Cantidad de bits que llegan invertidos, por unidad de tiempo.

- **Capacidad de un canal.** Es la máxima velocidad de transmisión posible
- **Ancho de banda de Nyquist**
  - Nyquist dedujo que la capacidad de un canal *sin ruido* es
 
$$C = 2B \lg(M)$$
    - C es la capacidad del canal.
    - B es el ancho de banda en Hz.
    - M es la cantidad de niveles de modulación.
    - (Para entender bien ésto, leer modulación multinivel, más adelante).
    - En particular, si tenemos sólo dos niveles, es  $C = 2B$ .
    - La idea es que si tenemos ancho de banda B, a lo sumo tenemos frecuencia proporcional a B. Esto significa que en hay B ciclos por unidad de tiempo, o sea 2B crestas de la onda por unidad de tiempo. Por cada cresta puedo codificar un bit (en el caso de M niveles, cada cresta puede codificar  $\lg(M)$  bits).
- **Teorema de Shannon-Hartley**
  - Shannon calculó la capacidad en un canal con ruido.
  - Relación señal-ruido (SNR):
 
$$\text{SNR}[\text{veces}] = (\text{potencia de la señal}) / (\text{potencia del ruido})$$
  - Expresión de SNR en dB (decibeles):
 
$$\text{SNR}[\text{dB}] = 10 \log_{10}(\text{SNR}[\text{veces}])$$
  - A mayor B, mayor es la frecuencia disponible y por ende mayor es C.
  - Por otro lado, a mayor B y frecuencia, mayor es la potencia del ruido el la señal y por ende menor es C.
  - Finalmente, a mayor potencia de la señal, menor impacto debería tener el ruido, y por ende mayor es C.
  - Shannon probó una formulita que captura todo esto:
 
$$C = B * \lg(1 + \text{SNR}[\text{veces}])$$
- Shannon + Nyquist
  - Acotando la capacidad de Nyquist, por la capacidad de Shannon, se llega a que
 
$$M \leq \sqrt{1 + \text{SNR}[\text{veces}]}$$
  - Esto nos dice que no podemos hacernos los vivos y aumentar la capacidad simplemente poniendo una cantidad de niveles arbitrariamente grande.
- **Teorema de Codificación de Canal de Shannon**

- Teorema muy grueso que dice que si transmitimos información a una tasa *menor* que la capacidad teórica del canal  $C = B * \lg(1 + \text{SNR})$ , entonces existe una codificación de la información que hace que la probabilidad de error tienda a 0. Por el contrario, si se transmite a una tasa *mayor*, no existe una tal codificación.
- En definitiva, este teorema nos dice que si queremos tener el error controlado, no podemos transmitir a velocidad mayor a C.

- **Medios de transmisión**

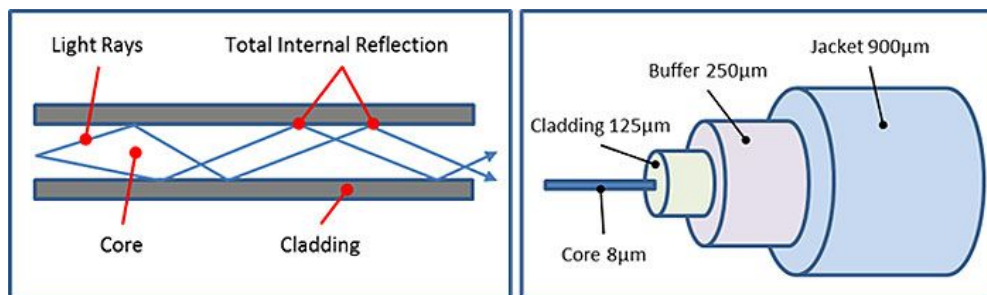
- Se clasifican en dos: con guía de onda y sin guía de onda (wireless).
- Par trenzado de cobre



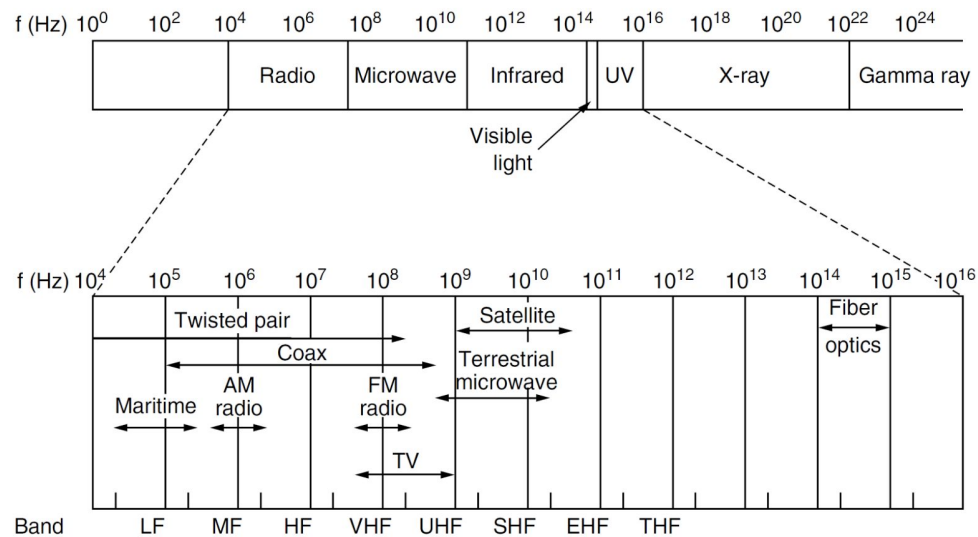
- Coaxial



- Fibra óptica



- La velocidad de propagación es  $\frac{2}{3} c$ .
- Sin guía de onda
  - Transmisión por radio, microondas (entre ellas, Wi-Fi), ondas infrarrojas, láser, Li-Fi.

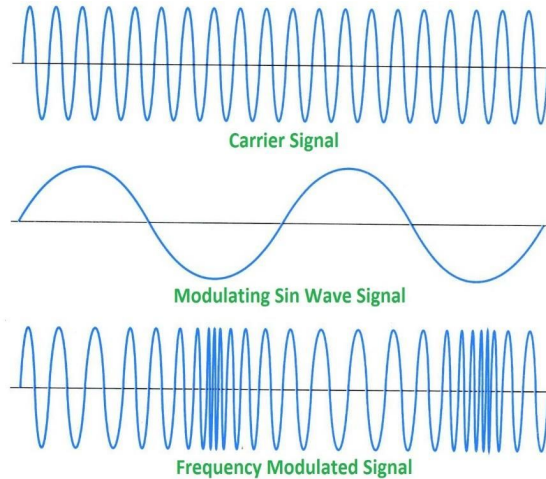


**Figure 2-10.** The electromagnetic spectrum and its uses for communication.

## • Modulación

- Codificación de un mensaje (datos analógicos o digitales) en una señal (analógica o digital).
- Un **módem** (MODulador-DEModulador) es un dispositivo que realiza modulación (y demodulación) de datos digitales en señales analógicas.
- Para esto, se tiene una señal base, llamada **señal portadora**, que es continuamente emitida. En cierto momento, se querrá enviar un mensaje, que se recibe en términos de una señal de entrada llamada **señal moduladora**. Para enviar el mensaje se varía alguna característica de la portadora (su amplitud, su frecuencia o su fase). Estas variaciones son las que codifican cada parte del mensaje. La señal resultante, que simbolizamos  $s(t)$ , se llama **señal modulada**.





- En otras palabras: la señal moduladora es el mensaje que queremos transmitir. Para transmitir ese mensaje, variamos la señal portadora y obtenemos una señal modulada, que si bien no es igual a la señal modulante, permite recuperar la moduladora el extremo receptor.
- Hay cuatro tipos de modulación, según el tipo de la señal moduladora y la portadora. El caso más común es el de moduladora digital y portadora analógica, i. e. mandar datos digitales a través de una señal analógica.

■ Técnicas:

- **Desplazamiento de Amplitud (ASK)**

- Los valores binarios se representan mediante dos amplitudes diferentes de la portadora.

$$s(t) = \begin{cases} A \cos(2 \pi f * t) & \text{si el bit es 1} \\ 0 & \text{si no} \end{cases}$$

- **Desplazamiento en Frecuencia (FSK)**

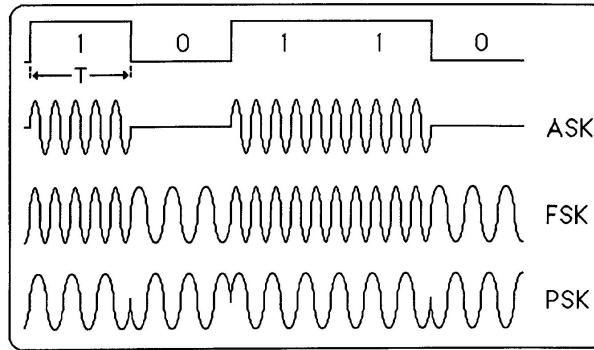
- Los valores binarios se representan mediante dos frecuencias diferentes de la portadora.

$$s(t) = \begin{cases} A \cos(2 \pi f_1 * t) & \text{si el bit es 1} \\ A \cos(2 \pi f_2 * t) & \text{si no} \end{cases}$$

- **Desplazamiento en Fase (PSK)**

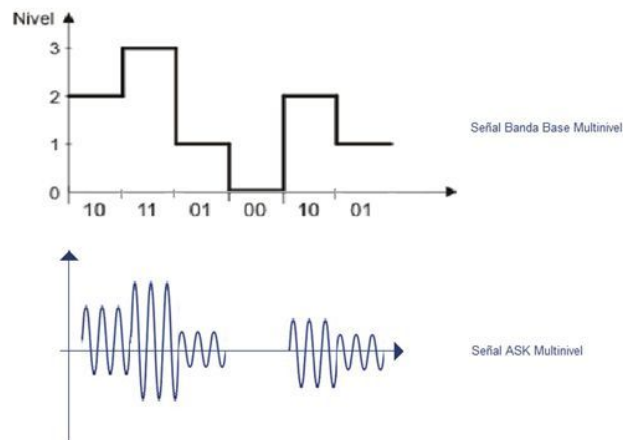
- Los valores binarios se representan mediante dos fases diferentes de la portadora.

$$s(t) = \begin{cases} A \cos(2 \pi f * t + \pi) & \text{si el bit es 1} \\ A \cos(2 \pi f * t + 0) & \text{si no} \end{cases}$$



## ■ Modulación multinivel

- Cada pedazo de la señal modulante representa varios bits.



ASK Multinivel

- Se puede lograr:
  - Aumentando el número de fases distintas (PSK).
  - Aumentando el número de amplitudes para cada fase (ASK-PSK).
- El número de fases o amplitudes distintas se denomina **nivel**. Cada símbolo de una modulación de  $M$  niveles codifica  $\lg(M)$  bits.

## ■ Modulación de Amplitud en Cuadratura (QAM)

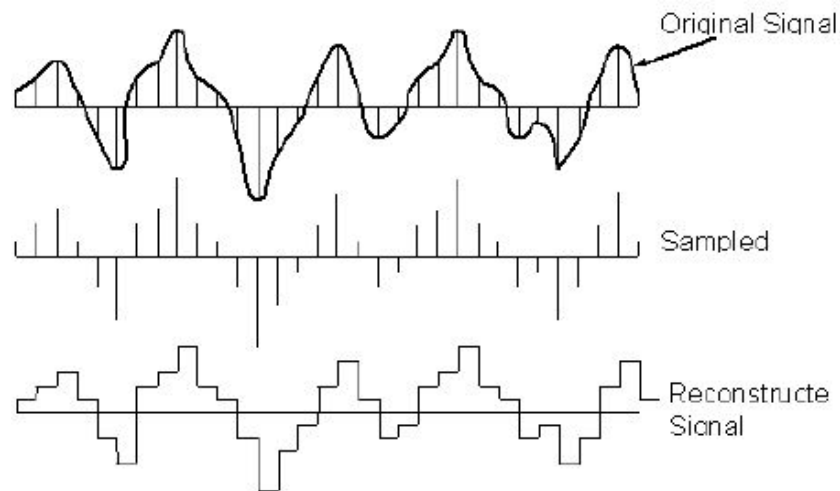
- Idea: para codificar más de un bit por elemento, utilizo varias ondas desfasadas.
- Si quiero codificar 2 bits por elemento, se utilizan dos ondas producto de una modulación ASK, desfasadas. Cada una codifica uno de dos bits consecutivos de la moduladora. Esto

se denomina 4 QAM (porque cada símbolo codifica 2 bits, o sea 4 estados posibles).

- Hay 8 QAM, 16 QAM, 64 QAM, 256 QAM.

- **Digitalización**

- A la modulación que produce una señal digital se la denomina digitalización. Este proceso transforma un mensaje (datos analógicos o digitales) en una señal digital.
- Un **códec** (COdificador-DECodificador) es un dispositivo que realiza modulación (y demodulación) de datos analógicos en señales digitales.
- Realiza una **conversión analógica-digital**. Esto consiste en tomar muestras (medir el valor) de la señal analógica en distintos puntos del tiempo, y obtener una discretización a partir de esas muestras.



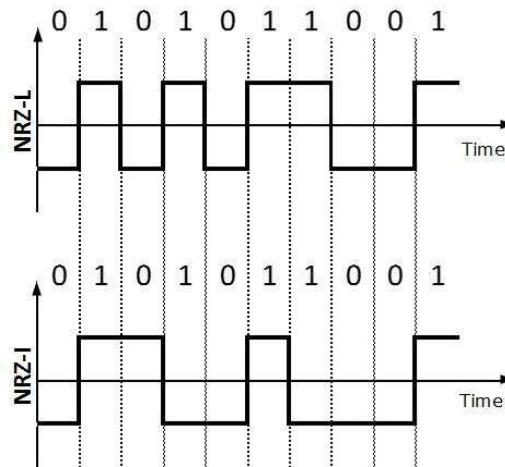
- ¿Con qué frecuencia debemos tomar una muestra de la señal, para poder reconstruirla?
- Recordemos que una onda analógica estaba compuesta de una suma de armónicos, cada uno con su propia frecuencia. El armónico más crítico es aquel de mayor frecuencia. Debemos muestrear con dicha frecuencia  $f_m$ . Como la frecuencia indica la cantidad de ciclos por segundo, y cada ciclo tiene dos crestas, debemos muestrear al menos  $2 * f_m$  veces.
- **Teorema del Muestreo (Nyquist)**. Si queremos reconstruir una señal de componente frecuencial máxima  $f_m$ , debemos muestrearla con una frecuencia  $f_s > 2 * f_m$ .
- Etapas de la conversión analógico-digital (**Pulse Code Modulation - PCM**):
  1. Se muestrea la señal al doble del ancho de banda de la misma.

2. Se cuantifican las muestras, aproximándolas mediante un número entero de  $n$  bits.

- En el caso de un mensaje digital codificado en una señal digital, los datos se transmiten codificando cada bit en un elemento de la señal digital. Esta señal digital se la puede presentar de varias formas.

- Non-return to Zero (NRZ)

- Los 0 se codifican con una tensión negativa.
- Los 1 se codifican con una tensión positiva.
- Problema: secuencias largas de 1 o 0 son difíciles de distinguir para el receptor. Si transmito 157 valores 1 seguidos, ¿cómo hace el receptor para reconocer que son exactamente 157? El receptor de alguna forma debería estar al tanto del clock del emisor (relojes sincronizados). Agregar una línea de comunicación que sólo transmita el clock es impráctico.



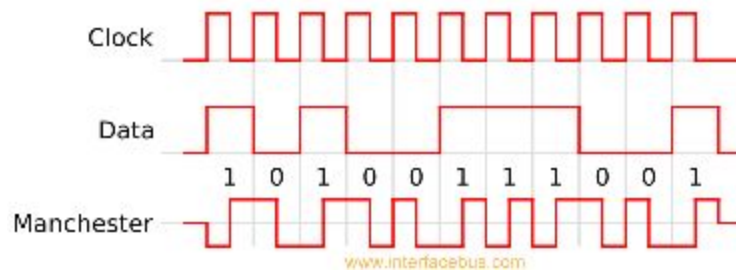
- Non-return to Zero Inverted (NRZI)

- En cada 1 se invierte la tensión. En los 0 no se hace nada.
- Soluciona el problema de muchos 1 consecutivos, pero no el de muchos 0.

- Manchester (bifase)

- Idea: durante el intervalo de cada bit, la señal debe alternar al menos una vez. Los dígitos se codifican en el medio de un intervalo. Los bordes del intervalo se utilizan para alternar la

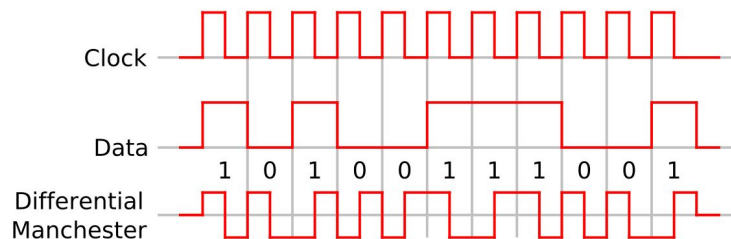
señal con el objetivo de sincronización.



- Los 1 se codifican alternando la señal de bajo a alto, en el medio de un intervalo.
- Los 0 se codifican alternando la señal de alto a bajo, en el medio de un intervalo.
- Se alterna al final de un intervalo, si es necesario, de modo tal que se pueda hacer el cambio de tensión indicado por el próximo bit.
- **Manchester = clock XOR datos**

■ Manchester (diferencial)

- Mezclamos la idea de Manchester, con la de NRZI.
- En cada 1 se alterna el sentido de las inversiones de tensión en el medio de los intervalos (si se invertía de tensión positiva a negativa, se pasa a alternar de negativa a positiva, y viceversa).



- Cualquiera de los dos Manchester soluciona el problema del sincronismo, pero tiene la mitad de la eficiencia que NRZ o NRZI, pues en cada ciclo de clock codifican 1 bit (y no 2, como los otros).
  - Alternativa: usar un esquema tipo NRZ, pero cambiar secuencias de varios bits iguales consecutivos, por alguna secuencia que el receptor pueda reconocer (e intercambiar por los bits originales), y produzca alternancia de la señal.
- Velocidad de modulación
  - **Velocidad de modulación.** Es el número de cambios de señal por unidad de tiempo, y se expresa en baudios (símbolos/seg).

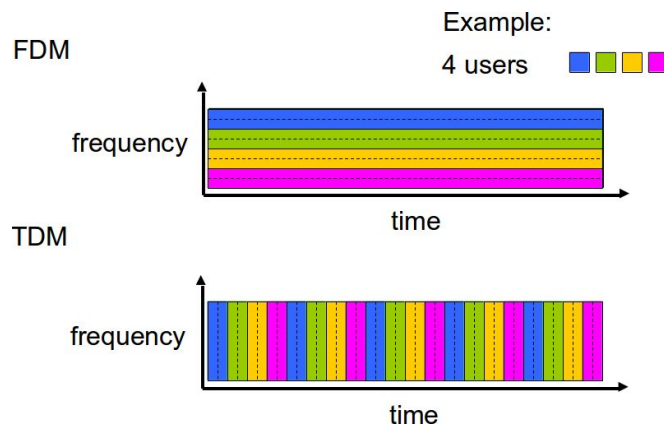
- **Velocidad de transmisión.** Es la cantidad de bits que se pueden codificar y emitir en una señal, por unidad de tiempo.

- En una modulación de M niveles, la velocidad de transmisión es

$$V_{tx} = V_{mod} * \lg(M)$$

- Multiplexación de señales sobre un canal

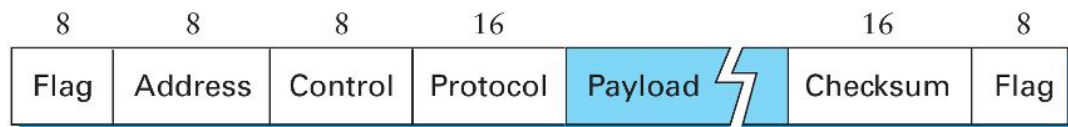
- Queremos poder enviar varias señales sobre un mismo canal, al mismo tiempo.
- Time Division Multiplexing (TDM)
  - Los usuarios del canal se turnan (round robin), obteniendo cada uno, periódicamente, el ancho de banda completo, por un período de tiempo acotado.
  - Sólo puede ser utilizado para datos digitales (imagino que porque en ellos se puede establecer un claro punto en el que cortar el turno de un usuario, sin romper todo).
  - Al parecer, se utiliza en telefonía.
  - Derrocha quantums si alguien no necesita utilizar el canal.
- Frequency Division Multiplexing (FDM)
  - Cada emisor opera en una porción del espectro de frecuencias.
  - Limita la cantidad de hosts de antemano (es difícil redistribuir el espectro de frecuencias).



- Wavelength Division Multiplexing (WDM)
  - Transmitimos varias ondas, cada una con distinta longitud de onda.
- Statistical Multiplexing
  - Paquete a paquete el switch decide quién manda (hay un límite superior de cantidad de paquetes consecutivos para mandar por host)
  - Se puede decidir utilizando FIFO, Round Robin, etc.

## Nivel de enlace: transmisión de datos

- Tenemos un canal de comunicación, sobre el cual podemos mandar señales.
- El canal está sujeto a ruido y fallas. Los datos enviados pueden ser alterados o llegar desordenados.
- Queremos enviar datos asegurando:
  - Confiabilidad. Los datos efectivamente llegan a destino.
  - Control de flujo. El emisor no sobrecarga de datos al receptor.
  - Control de errores. Detectar y corregir errores en la transmisión.
- Idea fundamental: encapsular los datos en frames (paquetitos).



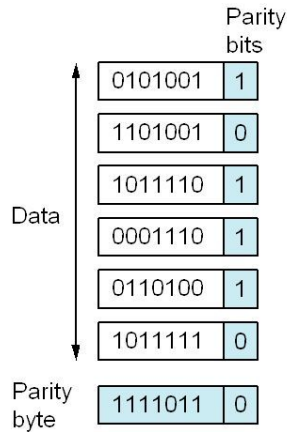
- Características de un frame
  - Un frame se compone de dos partes:
    - Header, que contiene metadata.
    - Payload, que contiene los datos que transporta el frame.
  - Debemos determinar dónde empieza y termina un frame. Técnicas:
    - Largo fijo
    - Largo especificado en un campo del header
    - Delimitadores de frame
      - Consiste en comenzar y terminar el frame con una secuencia de bits específica.
      - Problema: ¿qué hacemos si dicha secuencia aparece en algún otro lado del frame?
      - Solución: **bit-stuffing**. Por ejemplo, supongamos que el delimitador es el byte 01111110 (6 unos). Si al escribir el frame aparece la secuencia 011111 (5 unos), se *escapa* la secuencia con un 0. El receptor, al leer 5 unos consecutivos, sabe que si el siguiente bit es un 0, fue simplemente agregado y debe descartarse. Si en cambio es un 1 (es decir, tenemos 6 unos consecutivos) lo que suceda depende del siguiente bit. Si dicho bit es un 0, es el delimitador, y si es un 1, estamos ante un error pues el emisor nunca va a emitir la secuencia 01111111.

- Se incluye información en el frame para detectar y corregir errores.

Técnicas:

- Paridad

- Agregamos bits de paridad.
- Suele usarse un esquema bidimensional.



- Este esquema nos permite detectar hasta 3 bits erróneos y corregir hasta 1. (Detectar N y corregir M significa que si hay N o menos errores, podemos detectar que hay algún error, y que si hay M o menos errores, podemos determinar exactamente cuáles son).

- Checksum

- Se computa una suma de todos los datos.

- Cyclic Redundancy Check (CRC)

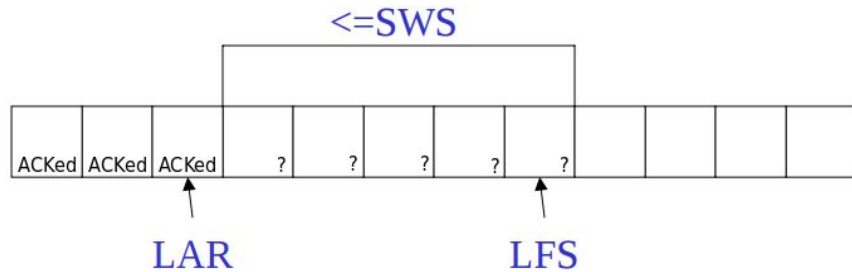
- Método basado en el cálculo de restos de ciertos polinomios. Más potente que Checksum pero más costoso de computar.

- Métodos de detección y corrección basados en codificaciones adecuadas

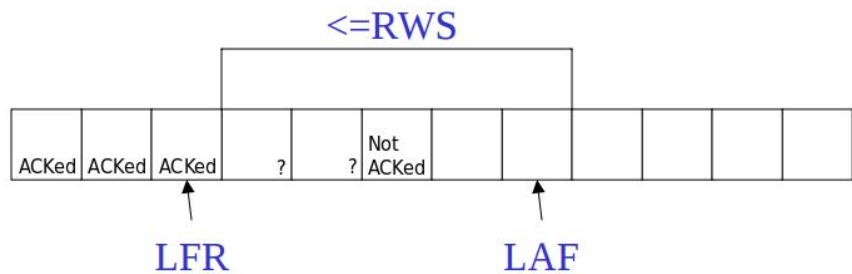
- Idea: dar una codificación de los datos, de modo tal que las codewords sean muy distintas entre sí. Al ser muy distintas, si alguna de ellas sufre alteraciones, las podremos detectar y quizás corregir.
- Llamemos  $d$  a la mínima distancia Hamming entre dos codewords del código (lo mínimo de distintas que tienen que ser).
- Sea  $e$  la cantidad de bits erróneos.
- Si  $e \leq d - 1$  podemos detectar, pues con  $d - 1$  cambios a una codeword la podemos convertir en otra, y por lo tanto vamos a detectar una secuencia de símbolos inválida.



- Si  $e \leq (d - 1) / 2$  podemos corregir, pues con  $(d - 1) / 2$  cambios a una codeword no llegamos “a mitad de camino” entre dos codewords válidas, y por lo tanto podemos determinar de qué codeword provenimos.
- Protocolos **ARQ** (Automatic Repeat reQuest)
  - Familia de protocolos que permiten asegurar confiabilidad y control de flujo.
  - Protocolo 1: **Stop & Wait**
    - Cada vez que el receptor recibe un frame exitosamente, emite un ACK al emisor.
    - El emisor no envía el siguiente frame hasta que recibe el ACK del receptor.
    - Si luego de determinado tiempo, el emisor no recibe el ACK, ocurre un timeout, y vuelve a enviarle el frame.
    - Es necesario secuenciar los paquetes (ponerles numerito), para evitar el problema de las **reencarnaciones**. Específicamente, necesitamos que el número de secuencia de los frames enviados alterne entre 0 y 1.
    - Problema: mientras el emisor espera a que el receptor conteste, no hace nada. Podría seguir enviando datos por el canal hasta recibir una respuesta.
  - Protocolo 2: **Sliding Window**
    - El emisor tiene una ventana de transmisión, y el receptor una ventana de recepción. Cada ventana indica la capacidad (de emisión o recepción, según corresponda) de buffering de cada extremo.
    - Todos los frames tienen un número de secuencia.
    - El emisor mantiene tres variables:
      - SWS: Sender Window Size. Cantidad de frames de su ventana. Es constante.
      - LAR: Last Acknowledged Received. Es el máximo frame para el que recibimos un ACK. Todos los frames  $\leq$  LAR fueron reconocidos.
      - LFS: Last Frame Sent. El último frame enviado.



- Invariante:  $LFS - LAR \leq SWS$
- El receptor mantiene tres variables:
  - RWS: Receiver Window Size. Cantidad de frames de su ventana.
  - LFR: Last Frame Received. Es el máximo frame para el que enviamos un ACK. Todos los frames  $\leq LFR$  fueron recibidos.
  - LAF: Largest Acceptable Frame. Es el máximo frame que estamos dispuestos a aceptar.



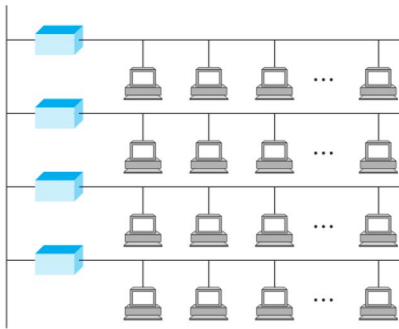
- Invariante:  $LAF - LFR \leq RWS$
- Notar que podría haber frames entre LFR y LAF que ya hemos recibido, pero que no hemos reconocido, porque aún no recibimos los del medio. En la figura, cuando recibamos los dos frames anteriores al *Not ACKed*, podremos enviar un ACK de ese frame.
- Semántica de los ACK
  - Los ACK son **acumulativos**.
  - Cuando el receptor envía un ACK para un frame con número de secuencia N, está confirmando que recibió *todas las frames hasta el N*. Luego, puede actualizar su LFR al frame N.

- Cuando el emisor recibe el ACK para el frame N, sabe que recibió todos los frames hasta el N. Luego, puede actualizar su LAR al frame N.
  - Todo esto implica que no hace falta mandar un ACK por cada frame recibido, sino que un ACK reconoce, también, todos los frames anteriores.
- Números de secuencia
  - Los números de secuencia son finitos, y eventualmente comenzamos a reutilizarlos. Debemos elegirlos con cuidado para evitar el problema de las reencarnaciones.
    - Supongamos que  $SWS = RWS = 5$ , y que secuenciamos con 5 números, de 0 a 4.
    - El emisor manda los primeros 5 frames, el receptor los recibe, pero los ACK se pierden en el camino. El receptor recibió los frames 0 a 4, y el siguiente frame en la secuencia es el 0 (es cíclico).
    - El emisor timeoutea y reenvía los frames 0 a 4. El receptor los recibe pero cree que son la nueva tanda de frames.
  - Se puede ver que para evitar este problema, debemos secuenciar con al menos  $SWS + RWS$  números.
- Relación entre SWS y RWS
  - No tiene sentido elegir  $RWS > SWS$ , porque de nada sirve que el receptor espere más frames que los que el emisor puede enviarle. El receptor nunca puede tener más de SWS frames no reconocidos.
  - Elegir  $SWS \geq RWS$  es perfectamente factible.
- Protocolo 2.5: **Go Back N**.
  - Es Sliding Window con  $RWS = 1$ .
  - El receptor no admite la recepción de frames fuera de orden.
  - Si el receptor recibe un frame fuera de orden, el inmediato siguiente, que ya fue recibido pero descartado, no será reconocido, y el emisor timeoutea. El emisor tiene que *volver a mandar* los N frames ya enviados partir de ese punto.
- Protocolo 3: **Sliding Window con Selective ACKs**
  - En lugar de emitir ACKs acumulativos, podemos emitir un ACK por cada frame individual que recibimos.
  - De esta forma, evitamos reenviar frames, por culpa de un único (o algunos pocos) frame intermedio que produjo timeout.

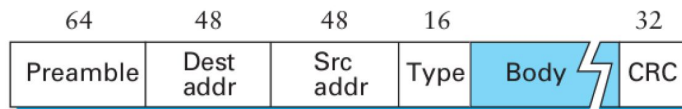
- Otros protocolos
  - **ACKs duplicados.** Cuando se recibe un frame nuevo, si están faltando frames intermedios, responder con un ACK duplicado (el último ACK emitido), para reafirmar la ausencia de esos frames intermedios.
  - **ACKs negativos.** Mandar un negative ACK advirtiéndole cuál es el frame que está faltando.
- Eficiencia de un protocolo
  - La eficiencia de un protocolo mide cuánto tiempo pasamos transmitiendo mientras estamos esperando por ACKs.
 
$$\eta_{\text{proto}} = T_{\text{tx}}(\text{window}) / \text{RTT}(\text{frame})$$
  - El SWS óptimo es aquel tal que  $\eta_{\text{proto}} = 1$ . Como  $T_{\text{tx}}(\text{window}) = |\text{window}| / V_{\text{tx}} = \text{SWS} * |\text{frame}| / V_{\text{tx}}$ , resulta que
 
$$\text{SWS} = (V_{\text{tx}} * \text{RTT}(\text{frame})) / |\text{frame}|$$
  - RTT y Delay
    - $\text{RTT}(\text{frame}) = 2 * \text{Delay}(\text{frame})$
    - $\text{Delay}(x) = T_{\text{tx}}(x) + T_{\text{prop}} + T_{\text{encol}} + T_{\text{proc}}$
    - $T_{\text{tx}}(x) = \text{tiempo de transmisión} = |x| / V_{\text{tx}}$
    - $T_{\text{prop}} = \text{tiempo de propagación de una onda sobre el enlace} = \text{longitud del enlace} / V_{\text{prop}}$
    - $T_{\text{encol}} = \text{tiempo que pasa un frame esperando en un buffer}$
    - $T_{\text{proc}} = \text{tiempo que insume el procesamiento del header, y de su enrutamiento}$
- **Logical Link Control (LLC)**
  - Es la capa que se encarga de ejecutar todos estos protocolos: el framing, el chequeo y corrección de errores y la ejecución de un protocolo ARQ.
  - Exporta, hacia los protocolos de capas superiores, el servicio de envío de un paquete a otro nodo.
  - Ofrece tres tipos de servicios:
    - Sin conexión y sin ACK. Es decir, no es confiable desde ningún punto de vista.
    - Sin conexión y con ACK. Sólo es confiable en el sentido de que cada paquete llega, aunque no tenemos control sobre el orden.
    - Orientado a conexión.

## Nivel de enlace: medios compartidos

- El medio que usamos para transmitir información puede estar compartido entre varios emisores de señales.
- Vimos algunas formas de multiplexar el canal, como TDM, FDM, WDM. Estos métodos requieren que los emisores se pongan de acuerdo de antemano, o que la emisión esté centralizada. No queremos estar sujetos a esas restricciones. El control de *acceso al medio debe ser descentralizado*.
- Por otra parte, al tener varios nodos conectados a una misma red, necesitamos poder identificar a cada uno. Necesitamos un *esquema de direccionamiento*.
- Tecnologías de medios compartidos clásicas:
  - Ethernet (802.3)



- Máximo 500m por tramo. Para evitar atenuación de la señal.
- Máximo 4 repetidores (aparatos que repiten y amplifican la señal). Por lo tanto, el enlace máximo mide 2500m.
- Mínimo 2,5m entre hosts.
- Se pueden usar distintos tipos de cables, que permiten distintas velocidades de transmisión.
- En una red Ethernet, debemos usar frames Ethernet:



- WiFi (802.11)
- Token-Ring (802.5)
- **Medium Access Control (MAC)**
  - Es la capa que se encarga de administrar el acceso al medio.
  - Es una capa intermedia entre LLC y el medio físico.
  - Su objetivo es maximizar el número de comunicaciones exitosas, y asegurar fairness (igualdad de oportunidades de acceso para todos los nodos).

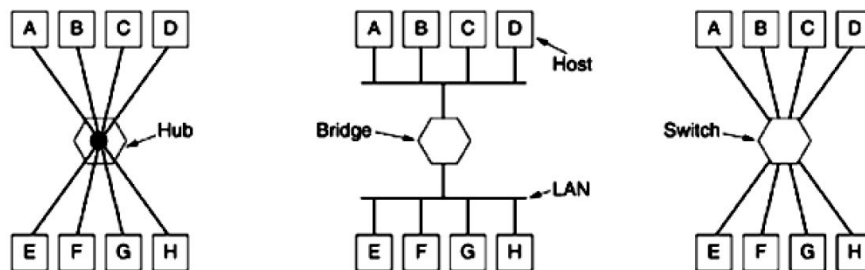
- La capa MAC se suele implementar con un protocolo de la familia **Carrier Sense Multiple Access (CSMA)**. El *Carrier Sense* es porque cada nodo sensa el medio (la señal portadora o *carrier*) antes de transmitir, y el *Multiple Access* porque, cualquier nodo puede decidir utilizar el medio.
- Uno de los sabores es **CSMA/CD** (la CD es de Collision Detection)
  - Cuando un host (un nodo) tiene datos para enviar, sensa el medio.
  - Si está libre, transmite.
  - Si está ocupado, espera a que se libere y transmite con probabilidad  $p$ . Se dice que, fijado el  $p$ , el algoritmo es  $p$ -persistente.
  - Durante el envío, se continúa sensando el medio para verificar si se produce una colisión. Si se detecta una colisión, se inserta una **jam sequence** (una señal que reafirma la interferencia) y se debe retransmitir.
    - Para determinar cuánto esperar hasta la retransmisión se usa **Exponential Backoff**.
    - Elegir un número entero al azar entre 0 y  $2^k - 1$ , con  $k$  la cantidad de intentos.
    - Esperar  $k$  veces el RTT máximo de la red, antes de sensar para retransmitir.
  - La comunicación half-duplex: si se está emitiendo una señal, no se puede recibir otra, porque esto provocaría una colisión.
- Ethernet:
  - Usa CSMA/CD y es 1-persistente.
  - Las direcciones de (interfaces de) hosts se denominan direcciones MAC.
  - Un host recibe frames que estén destinados a:
    - Su dirección.
    - La dirección broadcast (FF:FF:FF:FF:FF:FF).
    - Una dirección de multicast a la que está suscripto (i. e. que su adaptador Ethernet tiene configurado).
    - Cualquier frame, en caso de que esté en *modo promiscuo*.
  - La longitud mínima de un frame es 64B. ¿Por qué?
    - Un host sólo puede detectar una colisión mientras está transmitiendo una señal. Por lo tanto, debe permanecer suficiente tiempo para asegurarse de que ningún otro host ha comenzado a transmitir en el interín. En otras palabras, debe tener suficiente cantidad de datos para transmitir.
    - ¿Cuán largo debe ser un frame para asegurarnos de que hemos detectado una colisión? En el peor caso, un host comienza a transmitir justo antes que nuestros datos lleguen. Esto es, luego de

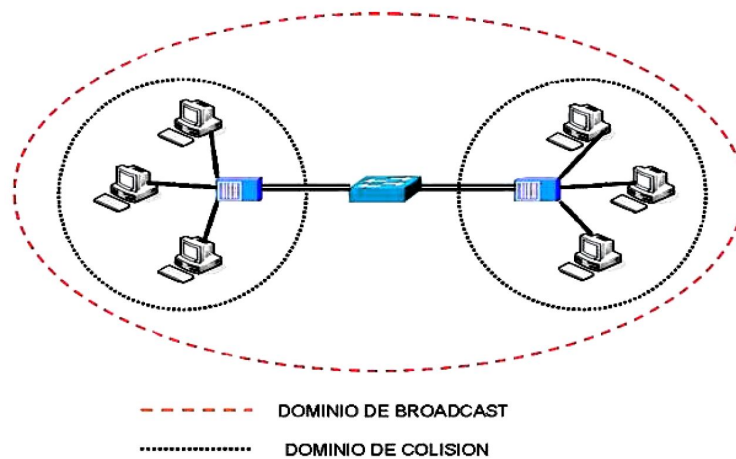
tiempo  $D_{\max}$ , el delay máximo de Ethernet. Luego de tiempo  $D_{\max}$  sentimos la señal del otro host. En definitiva, en el peor caso pasa tiempo  $2D_{\max}$  hasta que sentimos una colisión.

- La estimación es que  $D_{\max} = 25,6 \mu s$  en Ethernet 10Mbps (esto está directamente relacionado con la distancia máxima de un enlace entre dos hosts).
- Luego, queremos que  $T_{tx}(\text{frame}) \geq 2 * D_{\max}$  y de aquí se deduce que  $|\text{frame}| \geq 512b = 64B$ .

- **Hubs, bridges y switches**

- Problema: las tecnologías como Ethernet permiten una red de tamaño muy limitado. Queremos construir redes más grandes.
- Los hubs, bridges y switches son dispositivos que tienen varios puertos (conexiones con enlaces), cada uno conectando con una porción de la red distinta.
- **Hub**
  - Dispositivo que recibe una señal por un puerto y la distribuye sobre el resto. En otras palabras, es simplemente un repetidor.
  - Extiende el dominio de colisión. Por ende, no sirve para expandir el alcance máximo una tecnología de red.
- **Bridge/switch**
  - Un bridge es un dispositivo que recibe una señal por un puerto, y *la emite sobre el puerto correspondiente*. Esto es, tiene la suficiente lógica para decidir el puerto por el que debe emitir una señal.
  - Extiende el dominio de broadcast pero no el de colisión. Permite unir varias redes más pequeñas (LAN), creando una red más grande (Extended LAN), aunque el dominio de colisión de cada red pequeña se mantiene.
  - La diferencia entre bridge y switch no es conceptual. Muchas veces se llama switch a un bridge con múltiples puertos (más de dos).

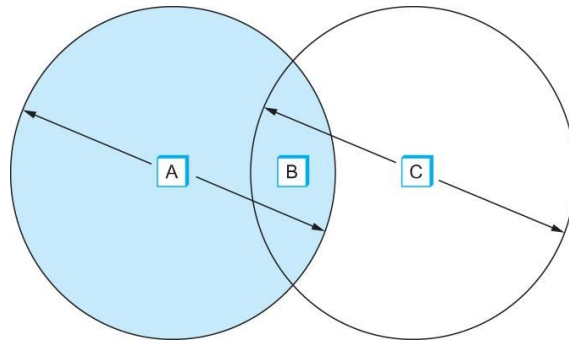




- Tecnologías inalámbricas
  - Como no hay un medio que guíe la señal, la señal está mucho más afectada por fuentes externas de ruido.
  - La señal se atenúa mucho más fácilmente viajando en el aire.
  - Como estamos compartiendo el medio (el aire) con otras señales electromagnéticas, debemos predefinir en qué ancho de banda debemos transmitir.
  - **Espectro disperso**
    - Cuando se transmiten señales en una banda no licenciada (es decir, libre, donde cualquiera puede transmitir) generalmente hay que compartir dicho espectro con otros dispositivos. Para prevenir interferencias se usa la técnica de espectro disperso.
    - Para transmitir una señal se utiliza un espectro de frecuencias mucho más grande que el estrictamente necesario para transmitirla.
    - Variamos regularmente la frecuencia a la que transmitimos la señal. La frecuencia a la que se transmite en cada momento está dada por una función, independiente de los datos transmitidos, conocida por ambos extremos de la comunicación.
  - Las comunicaciones inalámbricas tienen que cuidarse de varios problemas:

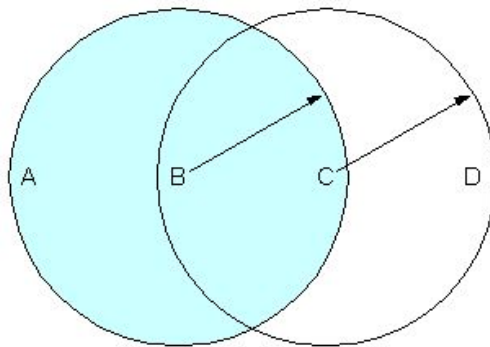


### ■ Problema de la estación oculta



- Tenemos tres hosts, A, B y C. Las circunferencias indican el alcance de las señales emitidas por A y C.
- A sensa el medio y no detecta otra señal. Transmite hacia B.
- C sensa el medio y no detecta la señal de A, porque C está fuera del rango de alcance de la señal de A. Transmite hacia B.
- Se produce una colisión de las señales en B, que pasa inadvertida por los dos emisores (encargados de retransmitir en caso de colisión).

### ■ Problema de la estación expuesta



- B sensa el medio y no detecta señales. Transmite hacia A.
- C quiere transmitir hacia D, pero cuando sensa el medio detecta la señal de B. Entonces no transmite.
- Sin embargo, si C emitiera su señal, sólo habría interferencia en la zona entre B y C, pero ni la señal de B ni la de C se verían comprometidas en la zona de los hosts receptores A y D, respectivamente.

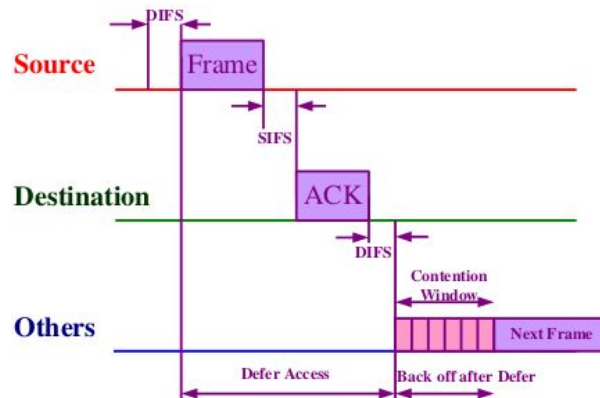
### ○ MAC en wireless

#### ■ ¿Podemos usar CSMA/CD? No. ¿Por qué?

- Mientras emitimos una señal no podemos sensar el medio para determinar si hay colisión.

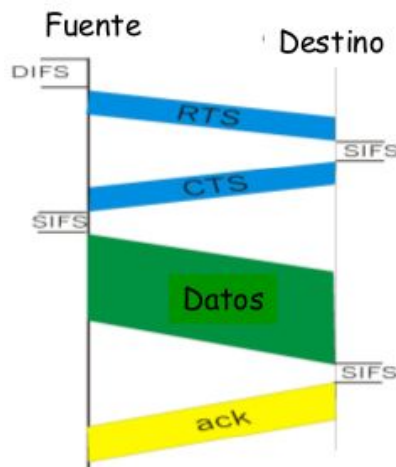
- Esto es porque la potencia de la señal emitida es, en general, mucho más grande que la potencia de una señal recibida (miles o millones de veces más potente).
- No todos los hosts pueden escuchar la señal de todos los otros.
- DCF (Distributed Coordination Function)
  - Es el mecanismo MAC utilizado en 802.11. Implementa **CSMA/CA** (CA de Collision Avoidance) con un algoritmo de exponential backoff.
  - Antes de transmitir, una estación determina el estado del medio.
  - Si está libre, espera un tiempo denominado DIFS (DCF InterFrame Space) y luego transmite. Al finalizar la transferencia, el receptor espera tiempo SIFS (Short InterFrame Space) y luego transmite un ACK, en caso de que haya recibido todos los datos exitosamente. Si no se transmite un ACK, el emisor ejecuta el algoritmo de exponential backoff.
  - Si está ocupado, se ejecuta el exponential backoff.
  - **Exponential backoff**
    - Se establece una **contention window** (ventana de slots de tiempo, análoga al backoff de CSMA/CD).
    - Se elige un **backoff counter** al azar, que determina la cantidad de slots de la ventana durante los que vamos a esperar.
    - Mientras el canal esté libre, se decrementa el backoff counter (en caso contrario se mantiene). Cuando el backoff counter llega a 0, se intenta transmitir el frame (comienza CSMA/CA de nuevo).
    - Si la transacción no es exitosa, se selecciona una contention window del doble de tamaño de la anterior.

## CSMA/CA Back off Algorithm



- **Mensajes RTS/CTS**

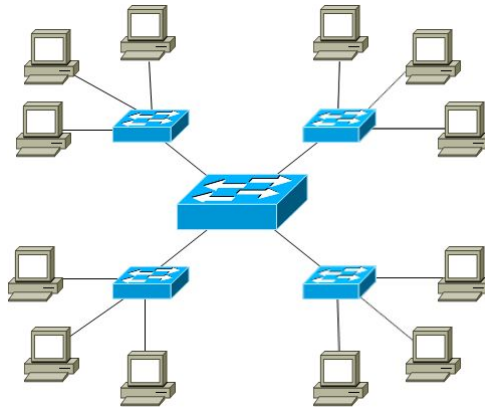
- Para evitar colisiones debido al problema de la estación oculta, CSMA/CA estipula un intercambio de mensajes previo al envío de los datos. El emisor envía un mensaje Ready To Send (RTS) y el receptor responde con un Clear To Send (CTS) en caso de estar de acuerdo.



## Nivel de enlace: switching

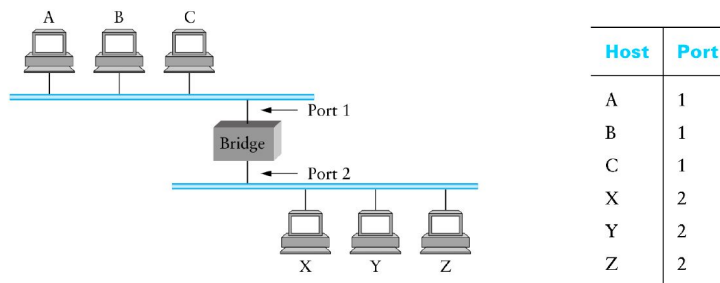
- Los switches permiten conectar varias LANs, formando una **Extended LAN**.
  - Las LANs conectadas pueden ser de distintos tipos de tecnologías.
  - Deben compartir el esquema de direccionamiento.
    - Para enviar un paquete hacia un host, debemos escribir la dirección de ese destino en un paquete de la tecnología del emisor.

- En cada switch que cruzan los datos, la tecnología de red podría cambiar, y por lo tanto podríamos tener que reescribir las direcciones en un nuevo paquete.
- La tarea de un switch consiste en recibir un paquete, y **forwardearlo** a la red correspondiente, i. e. por alguna de sus interfaces (puertos). El forwardeo lo hace en función de la dirección de destino. ¿Cómo hace el switch para determinar a qué red debe forwardearlo?



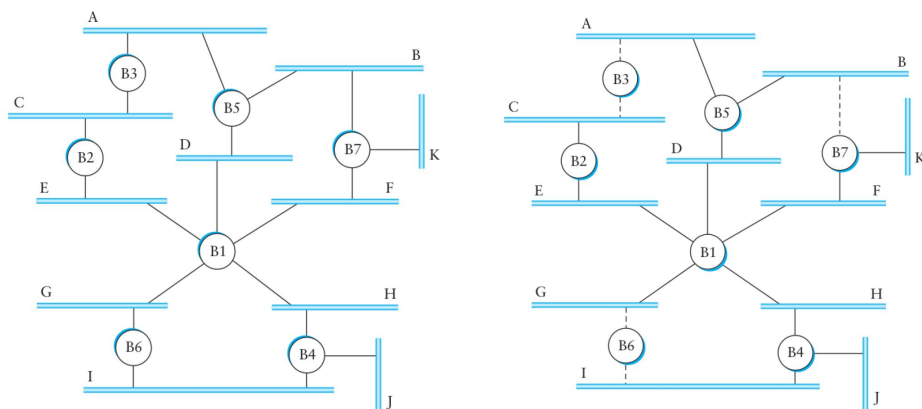
### ● Learning Bridges

- Cada vez que un switch recibe un frame con una cierta dirección de origen, asocia la interfaz por la que entró el frame con dicha dirección.



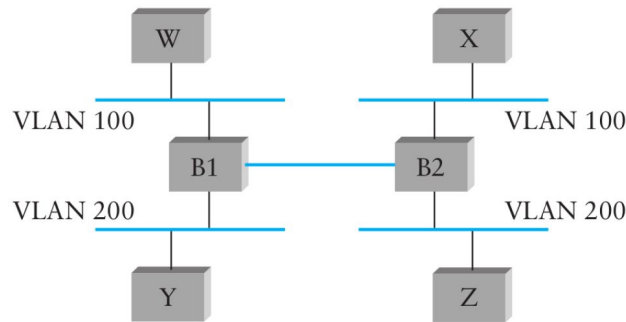
- Van construyendo una tabla, llamada **tabla de forwarding**.
- Problema: las topologías con ciclos. Producen forwardeos ad-infinitum.
- **Spanning Tree Protocol**
  - Protocolo que elimina ciclos en una red con switches.
  - Idea: tomar un árbol generador del grafo de la red.
  - Protocolo:
    - Todos los switches tienen un id. El de id más chico será el switch **root**. El árbol generador que se construirá es un árbol de caminos mínimos, en cantidad de hops, con raíz en el root.
    - En cada momento, cada switch mantiene la siguiente información:

- Cuál es el switch root.
  - Alguna interfaz que está en un camino mínimo hacia el root. En otras palabras, una interfaz con distancia mínima al root. Esta interfaz se denomina **root port**.
  - Algunas interfaces que estén en un camino mínimo entre la LAN de esa interfaz y el root. En otras palabras, por cada LAN se elegirá una interfaz de un switch que tenga distancia mínima al root. Estas interfaces se denominan **designated port**.
- Los puertos root y designados van variando a lo largo del algoritmo. Cuando se alcanza el resultado, los puertos que no son ni designados ni root, se denominan **blocked port**.
  - Inicialmente, los switches no tienen información sobre el resto de los switches y la topología. Creen que son el root.
  - Los switches propagan la información que tienen, a sus vecinos. Lo hacen a través de paquetes BPDU (Bridge Protocol Data Units).
  - Un BPDU es una terna de valores:
    - El id del que está enviando el mensaje.
    - El id del root según el que envía el mensaje.
    - La distancia, en hops, desde el que envía el mensaje hasta el root.
  - Un switch actualiza su información al recibir un BPDU (switch\_id, root\_id, dist) si:
    - El root\_id es menor que el que conoce.
    - El root\_id es igual, pero dist es más chica.
    - El root\_id es igual y la dist también, pero switch\_id es más chico.



- Virtual LANs (VLAN)

- El broadcast no escala. En consecuencia, las Extended LANs (que extienden el dominio de broadcast) no escalan.
- Una solución es crear Virtual LANs.
- Las VLANs permiten particionar a una LAN (o Extended LAN) en varias LANs diferentes y conectarlas. Cada VLAN es un dominio de broadcast distinto.



- Por ejemplo, en la figura tenemos la VLAN 100 y VLAN 200. Los bridges B1 y B2 se configuran de modo tal que si el host W emite un broadcast, esos datos son forwardados por B1 a B2, y éste lo forwardea por la interfaz de X. Estas decisiones las toman utilizando un VLAN id, que es incluido en el frame. Además, tanto B1 como B2 saben sobre qué interfaz está cada VLAN.
- **Switching orientado y no orientado a conexión**
  - Una *conexión* es una ruta predefinida (una secuencia de redes y switches) por los que pasa un paquete para llegar a su destino.
  - **Orientado a conexión (Virtual Circuits)**
    - Cuando un host quiere enviar datos a otro, se realiza una fase de conexión. Al terminar la transferencia, debe cerrarse la conexión.
    - En esta fase, se define un circuito, denominado **circuito virtual (VC)**, por la cual viajarán los paquetes entre los hosts. Cada switch intermedio aprende hacia dónde debe conmutar los paquetes correspondientes a dicha comunicación, para que sigan esa ruta. Para conmutar utiliza una **tabla VC**.
    - Una tabla VC tiene 4 columnas:

Puerto de entrada	VCI de entrada	Puerto de salida	VCI de salida
2	5	1	11

VCI es Virtual Circuit Identifier, y es un numerito que le asigna cada switch al circuito. Por ejemplo, en la tabla presentada, cuando el

dueño de la tabla reciba un paquete con VCI 5 por el puerto 2, lo va a forwardear por el puerto 1 con VCI 11.

- Este esquema de switching se denomina **conmutación de circuitos**.
- **Sin conexión (Datagramas)**
  - Un host puede enviar un paquete sin pasar por una fase de conexión.
  - Cada switch conmuta cada paquete individualmente, según su destino. Cada vez que un paquete llega a un switch, éste decide, en ese momento, hacia dónde forwardear. No hay una ruta predefinida. Para conmutar utiliza la tabla de forwarding.
  - Lo anterior implica que cada paquete debe llevar toda la información necesaria para alcanzar su destino (como mínimo la dirección de destino).
  - Este esquema se denomina **conmutación de paquetes** u orientado a datagramas.
- Con conexión vs. sin conexión
  - Con conexión tiene que usar al menos un RTT para establecer una conexión. Sin conexión puede comenzar a enviar datos inmediatamente.
  - Sin conexión tiene que enviar todos sus paquetes con toda la información requerida para llegar a destino (como mínimo su dirección). Con conexión demanda mucho menos overhead por paquete para switchear el paquete, sólo se necesita una forma de identificar el circuito a seguir.
  - Con conexión fija un camino, y no lo puede cambiar. Sin conexión se puede adaptar a cambios de la topología de red, como por ejemplo debido a nodos que fallan.
  - Con conexión tenemos la certeza de que la red puede entregar los datos y de que el receptor está listo para recibirlos.
  - Con conexión nos permite reservar espacio de buffer en cada uno de los switches intermedios. Esto permite dar garantías de calidad de servicio.

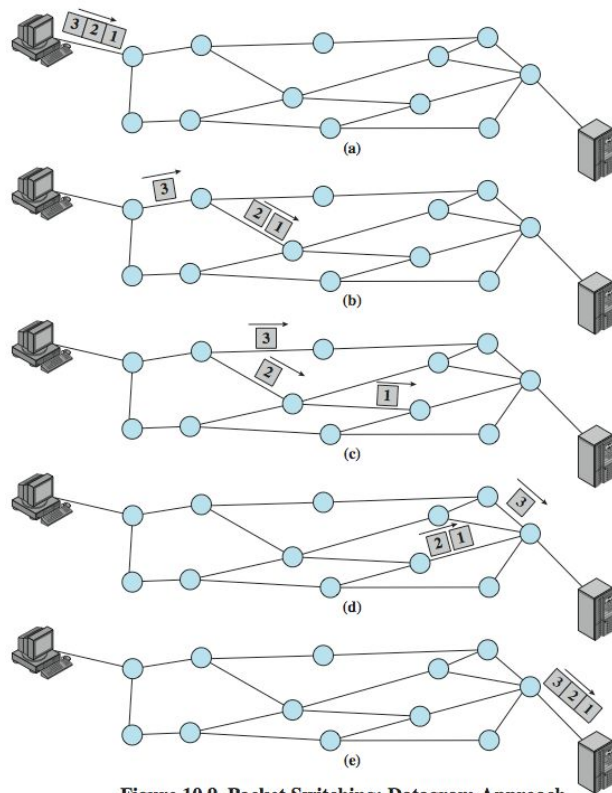


Figure 10.9 Packet Switching: Datagram Approach

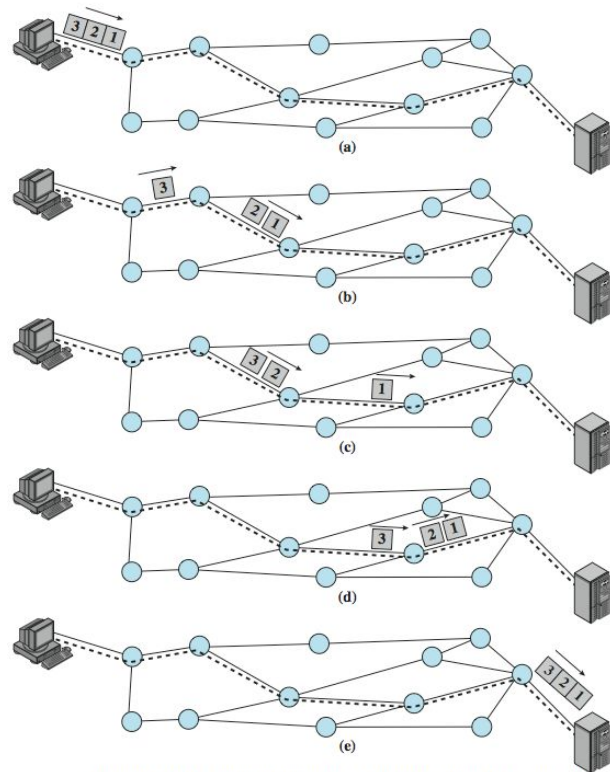


Figure 10.10 Packet Switching: Virtual-Circuit Approach

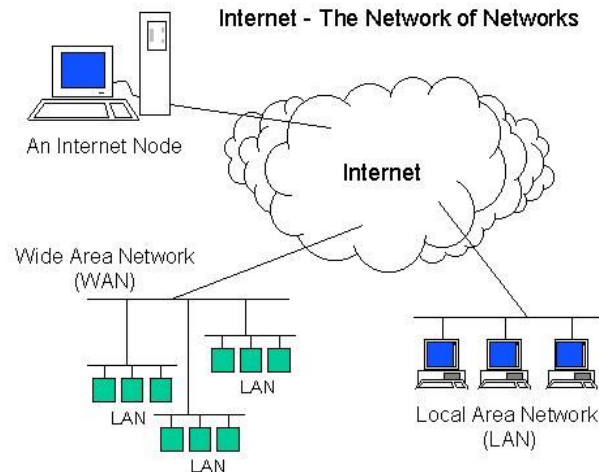
- Métodos de establecimiento de una conexión
  - Conexión Permanente (PVC)
    - El administrador de red va switch por switch, agregando la conexión a las tablas VC.
  - Conexión por Solicitud (SVC)
    - Cuando un host desea enviarle datos a otro, manda un mensaje de solicitud de conexión por la red. La solicitud viaja switch por switch, hasta llegar al destino, y luego vuelve un ACK, siguiendo el camino inverso, hasta volver al emisor. Cada vez que un switch recibe una solicitud de VC, crea una nueva entrada en su tabla VC con un VCI de entrada nuevo. Cuando le vuelve el ACK de la solicitud, escribe el VCI de salida (que es el VCI que viene con el ACK desde el switch siguiente), y le pasa el VCI de entrada al switch anterior en el ACK.
    - La finalización de la conexión en cada caso es análoga.
- Conmutación **Source Routing**
  - Toda la información que se necesita para conmutar los paquetes es proporcionada por el nodo de origen.



- Esencialmente, el host emisor incluye el camino completo que debe seguir el paquete para llegar a destino.
- Variantes:
  - Rotación. Inicialmente está escrita la ruta  $N_1, N_2, \dots, N_k$ . El primer switch forwarda a  $N_k$  y rota la ruta a  $N_k, N_1, N_2, \dots, N_{k-1}$ . El segundo switch (el nodo  $N_k$ ) forwarda a  $N_{k-1}$  y rota. Así sucesivamente.
  - Striping. En lugar de rotar, vamos cortando el nodo a la cabeza de la ruta.
  - Puntero. El paquete tiene un puntero al siguiente nodo al que hay que forwardear.

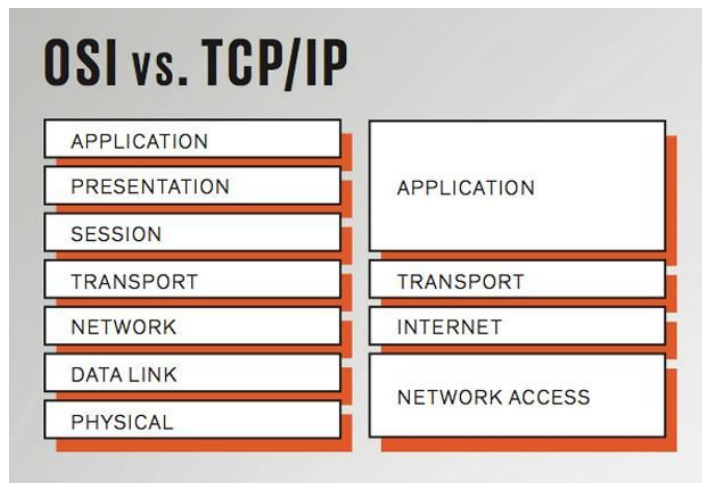
## Nivel de red: arquitectura

- Internet es una red compuesta de muchas otras redes más pequeñas.

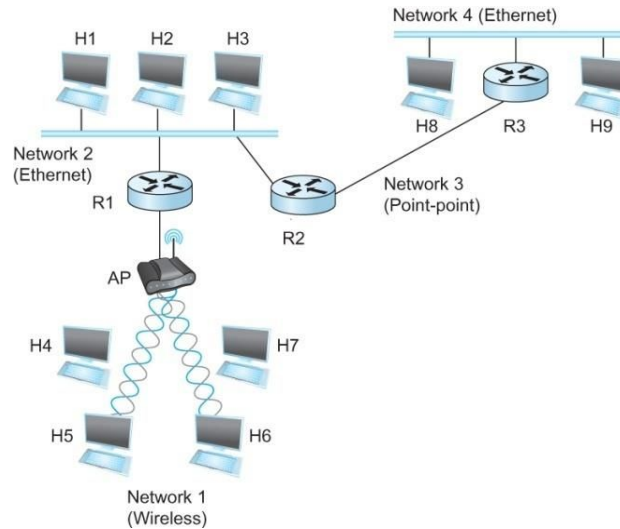


- Cada una de esas redes puede usar una tecnología distinta. En la sección anterior, vimos cómo es la comunicación dentro de cada una de esas redes más pequeñas. Los protocolos que permiten esa comunicación son los de **capa 2**, la **capa de enlace**.
- Queremos que dos hosts, en dos redes distintas, corriendo sobre tecnologías posiblemente distintas, puedan comunicarse.
- Necesitamos agregar una capa arriba, que permita esta comunicación inter-redes.
- Modelos de capas
  - Las capas son una forma de pensar la comunicación.
  - OSI es un modelo de capas que se planteó en los inicios de Internet. Tiene una gran cantidad de capas. Las más notables son:

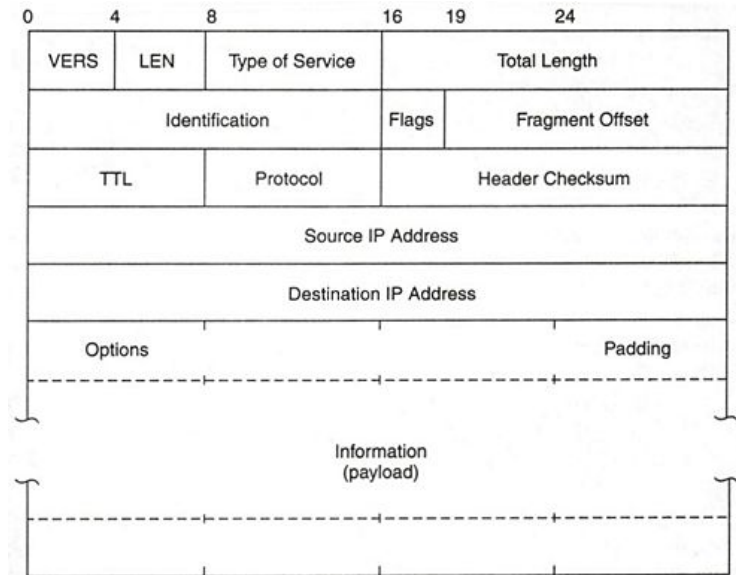
- Física: cómo mandar las señales a través del medio. De esto se los adaptadores de red de los dispositivos.
- Enlace: envío de datos en una red switchada (o sea, de escala pequeña-mediana). Protocolos como MAC, ARQ, STP o Learning Bridge, están todos en esta capa.
- Red: envío de datos entre redes. Es lo que vamos a tratar en esta sección.
- Capas superiores: más adelante.
- El modelo de capas TCP/IP es una simplificación del anterior, y es el que se usa en la práctica.



- Como queremos comunicar hosts de distintas redes, necesitamos una forma universal de identificar a cada uno de ellos. Es decir, un sistema de direccionamiento independiente de las direcciones de cada tecnología de red particular.
- ¿Cómo interactúan las capas?
  - Cuando un nodo desea enviar datos a otro, que está en otra red, crea un paquete de la capa de red. Introduce su dirección y la dirección del destino.



- Para llegar a ese destino, hay que pasar por otros nodos intermedios. Entre cada par de nodos intermedios se usa una tecnología de red distinta, con lo cual en cada salto hay que usar un protocolo de capa de enlace. Por lo tanto, entre cada par de nodos, el paquete del protocolo de red, va a viajar encapsulado (será el valor del campo de datos) de un paquete de capa de enlace.
- **IP (Internet Protocol)** es un protocolo de capa de red. Esto significa que su objetivo es comunicar distintas redes.
  - Como se debe adaptar a muchas redes de distintas tecnologías, debe ser suficientemente general.
  - Es orientado a datagramas (sin conexión).
  - Es best-effort. Esto significa que no es confiable.
    - Los paquetes se pueden perder.
    - Pueden llegar fuera de orden.
    - Pueden llegar repetidos.
    - No hay un límite preestablecido para el tiempo de entrega.
  - Formato de paquete:



■ Descripción de algunos campos:

- **TTL (Time To Live)**. Es un contador de hops. Es regresivo. El emisor lo setea en cierto número, y en cada salto se decrementa en una unidad. Cuando llega a 0, el paquete se descarta.
- Checksum. Del header, sin los datos.
- Dirección fuente y destino. Las famosas IPs. Son de 32 bits.
- **Protocol**. Indica el protocolo de nivel superior al que corresponden los datos encapsulados. Este campo es en análogo al campo Type del paquete Ethernet. En ese caso, hay un valor de Type que indica que el contenido es un paquete IP.
- Identification, Flags y Fragment Offset. Utilizados para fragmentar paquetes (explicado a continuación).

○ **Fragmentación de paquetes**

- Cada tecnología de red admite distintos tamaño máximo de paquete. Esto se conoce como **MTU (Maximum Transmission Unit)**. Por ejemplo, Ethernet tiene un MTU de 1500B.
- IP se debe poder adaptar a todos los MTU. Para lograr esto, debe ser capaz de fragmentar sus paquetes si son muy grandes.
- Es necesario fragmentar cuando  

$$\text{tamaño(paquete IP)} + \text{tamaño(header protocolo de enlace)} > \text{MTU}$$
- El reensamblado se realiza en el host destino. Es decir, si en cierto enlace intermedio se decide fragmentar, el paquete viajará fragmentado hasta el final.

- Las direcciones MAC eran flat: no daban ninguna pista sobre la localización de un host. No necesitábamos esa información, porque la red era suficientemente chica como para que los switches aprendieran cómo conmutar hacia cada dirección MAC. En Internet, no podemos mantener información de forwarding sobre todas las direcciones.

- Así como los switches son los aparatos que entienden los protocolos de nivel de enlace, los **routers** son los que entienden los protocolos de nivel de red.
- Un router toma un paquete IP, y debe forwardearlos por las interfaces correspondientes. Utiliza una **tabla de forwarding**.
- Dada la dirección IP del host destino, el router hace lo siguiente:

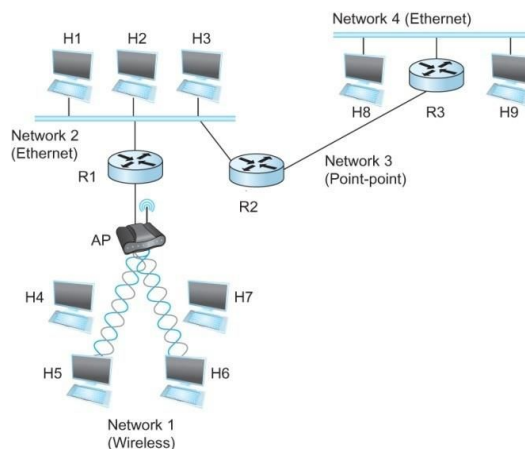
- Si el router está directamente conectado a la red destino, forwardear el paquete al host.
- Si no, forwardear el paquete a un router.

```

if (NetworkNum destino = NetworkNum de algunas de mis
interfaces) then
    enviar datagrama al destino por esa interfaz
else
    if (NetworkNum del destino está en mi forwarding table)
    then
        enviar datagrama al NextHop router
    else
        enviar datagrama al default router
    end if
end if

```

- En la siguiente red, el router R2 tiene la siguiente tabla de forwarding (la interfaz izquierda es la 1, la derecha es la 0):



NetworkNum	NextHop
1	R1
2	Interface 1
3	Interface 0
4	R3

- La tabla puede definir un default router, que es el router al que le enviamos paquetes cuando no sabemos cómo forwardear.
- **IPv6**
  - Versión 6 de IP. Las direcciones son de 16B.
- **ATM (Asynchronous Transfer Mode)**
  - Es una tecnología de red basada en circuitos virtuales.
  - Todos los paquetes tienen el mismo tamaño fijo (53B). Esto hace que el procesamiento y forwarding de paquetes sea mega rápido.

# Nivel de red: routing

- El problema de **routing**
  - Consiste en construir una tabla de forwarding que permita mandar paquetes por caminos de costo mínimo en la red.
  - Es un problema de optimización sobre grafos. Los nodos son los routers de la red, y las aristas son los enlaces que los conectan. Cada enlace tiene un costo asociado, el costo de enviar un paquete por ese enlace. No es evidente cómo determinar el costo de cada enlace.
  - Un algoritmo de ruteo calcula y utiliza, a lo largo de su ejecución, una tabla de ruteo. A partir de esta tabla se calcula la de forwarding.
  - Tipos de ruteo:
    - Estático: configuración manual. No responde a cambios de topología. No escala.
    - Dinámico: configuración automática y adaptativa.
- Clases de protocolos de ruteo
  - **Ruteo Interno (IGP o Interior Gateway Protocols)**
    - Su dominio de ruteo es dentro de un **Sistema Autónomo (AS)**. Un AS es un grupo de redes IP que poseen una política de ruteo independiente. Por ejemplo, la red de un **ISP (Internet Service Provider)**.
  - **Ruteo Externo (EGP o Exterior Gateway Protocols)**
    - Usados en redes de varios AS.
- IGP 1: **RIP (Routing Information Protocol)**
  - Implementa un algoritmo llamado **Distance Vector**. Es un Bellman-Ford distribuido.
  - Cada router construye una tabla de ruteo con la siguiente información:

Destination	Cost	NextHop
-------------	------	---------

- Una fila (R, C, N) indica que se puede llegar al router R, con costo C (que puede ser, por ejemplo, cantidad de hops o delay), y para realizar esa distancia hay que ir al router N.
- Además, cada router mantiene un vector con las distancia cada uno de los routers de la red. Inicialmente, cada router solo sabe cómo llegar a sus vecinos. Las distancias a todos los demás es infinito.
- Algoritmo:
  - Periódicamente (cada cierta cantidad de segundos) o cuando su vector de distancias cambia, cada router envía su vector de distancias a sus vecinos.

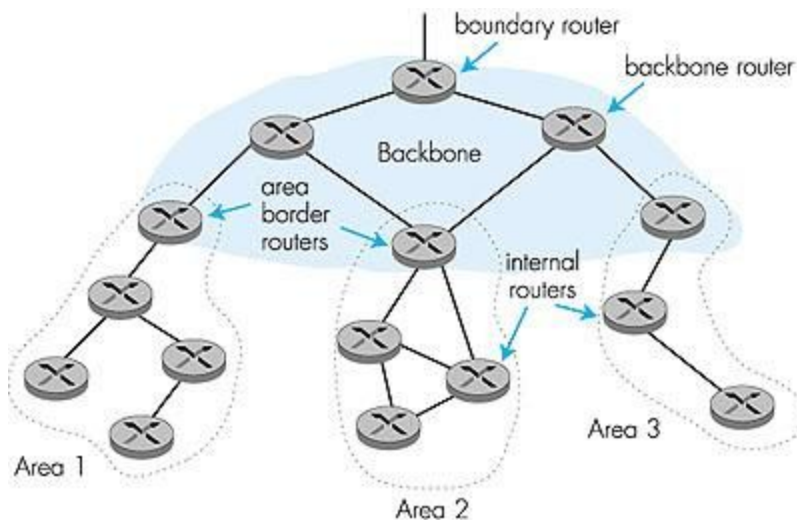
- Cada vez que un router recibe un vector de distancias, se actualiza la tabla de ruteo y el vector de distancias. Un router R1, al recibir el vector de un vecino R2, una entrada (X, D, R3) en R1 se actualiza si:
  - $d(R2, X) + 1 < D$
  - $R2 = R3$  (en este caso, incluso se actualiza si  $d(R2, X) + 1 > D$ , pues esto podría significar un cambio en la topología)
- Cuando la comunicación con un vecino produce timeout, se ponen en infinito todas las distancias que lo tengan como NextHop.
- Problema: **count to infinity**
  - Supongamos que tenemos una red A -- B -- C, y que las distancias ya están calculadas. B sabe que puede llegar a A en 1 hop.
  - El nodo A se cae. B nota esto, y pone en infinito su distancia a A. C aún cree que su distancia a A es 2.
  - C transmite  $d(C, A) = 2$  a B. B actualiza a  $2 + 1 = 3$  su distancia a A. Esto es erróneo porque el camino de C a A pasaba por B.
  - B transmite  $d(B, A) = 3$  a C. Si bien  $d(C, A) < 3$ , dicha distancia desde C se realizaba con NextHop = B. Entonces se actualiza  $d(C, A) = 3 + 1$ .
  - Este ciclo se repite infinitamente.
  - El problema es que B actualizó su distancia  $d(B, A)$ , a través de  $d(C, A)$ , pese a que la distancia de C a A se realizaba pasando por B.
  - Soluciones:
    - Poner un número máximo para las distancias, que represente infinito. Por ejemplo, 16. Entonces, en el ejemplo anterior, el ciclo de incrementos terminaría en 16.
    - **Split horizon**. Evita que un nodo X envíe la información de distancias a otro nodo Y, que fue obtenida de X. En otras palabras, si  $d(X, Z)$  se realiza pasando por Y, entonces X no le envía esa distancia a Y.
    - **Split horizon with poison reverse**. Análogo a lo anterior, pero en lugar de no transmitirle esa distancia  $d(X, Z)$ , se transmite  $d(X, Z) = \text{infinito}$ .
    - Estas últimas dos técnicas funcionan sólo cuando el lazo involucra a dos nodos (como en el ejemplo de A, B y C).

- IGP 2: **OSPF (Open Shortest Path First)**



- Implementa un algoritmo llamado **Link State**. Es un Dijkstra desde cada router.
- Algoritmo:
  - Cada router descubre sus routers vecinos y mide el costo del enlace con cada uno de ellos.
  - Una vez que cada router conoce todos esos costos, crea un paquete con esta información y se la envía a todos los nodos de la red, i. e. se hace un **flooding** del paquete.
  - Eventualmente, todos los routers tienen la información completa de la topología de la red y los costos entre cada par de routers. Cada uno ejecuta una versión del algoritmo de Dijkstra para calcular las distancias desde allí hacia el resto de los routers.
- El paquete con la información que envía cada router se llama **Link State Packet (LSP)**. Contiene:
  - Id del router que lo creó.
  - Costo del enlace a cada vecino directamente conectado.
  - Número de secuencia (SEQNO). Es necesario secuenciar los paquetes, pues como circularán por toda la red, es necesario poder distinguirlos la última versión en caso de que haya más de una dando vuelta.
  - TTL. Es necesario para saber cuándo eliminar un paquete de la red, y además se usa para ir *envejeciendo* la información.
- **Reliable flooding:**
  - Hacer un flooding correctamente es difícil. Entre las dificultades de este proceso están el poder determinar la última versión de la información circulando, y determinar cuándo un paquete debe ser descartado (si nunca se descartara un paquete, circularía eternamente por la red).
  - Para garantizar un flooding confiable, cada router hace lo siguiente:
    - Almacena el LSP más reciente de cada nodo.
    - Cada vez que recibe un LSP, lo redistribuye a todos sus vecinos, excepto a quien me lo envió, decrementándole primero el TTL.
    - Periódicamente genera un LSP propio nuevo, incrementando el SEQNO.
    - Periódicamente decrementa el TTL de cada LSP. Esto produce que eventualmente se descarte la información vieja. Cuando un TTL llega a 0 se hace ACK de ese LSP.

- Los paquetes OSPF pueden viajar adentro de IP. Hay un número para el campo Protocol de IP estipulado para OSPF.
- Vector Distance vs. Link State
  - Vector Distance transmite toda la información de distancias a sus vecinos.
    - Si hay N nodos, cada nodo usa  $O(N)$  memoria.
    - Converge lento.
  - Link State transmite la información de sus vecinos a todos los nodos.
    - Si hay N nodos y M ejes, cada nodo usa  $O(N + M)$  memoria. Si la red es muy densa, esto es mucha memoria.
    - No genera demasiado tráfico y responde rápido a cambios de la topología.
    - Se estabiliza rápido.
- IGP 3: OSPF jerárquico



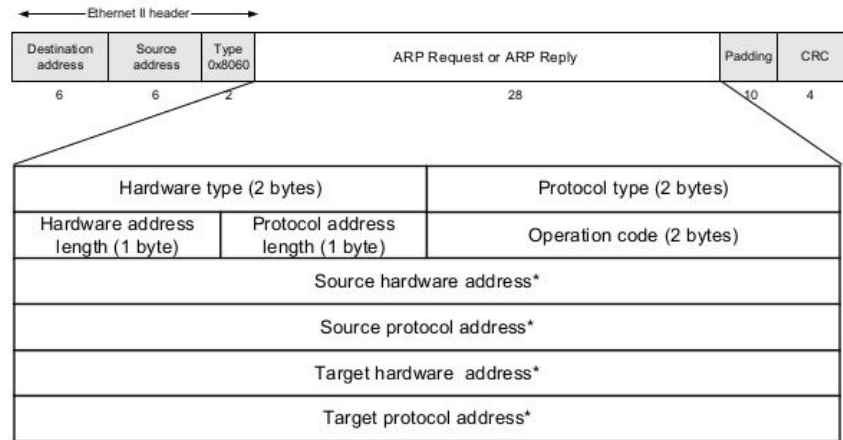
- Se ejecuta OSPF limitándose al área troncal.
- Los routers frontera de área (los de la frontera con cada área) resumen las distancias a los routers de las redes de su área.
- Características de los EGP
  - Los protocolos EGP se preocupan por alcanzar nodos, no por optimizar rutas.
  - Mensajes del protocolo:
    - Adquisición de vecinos: un router quiere que otro sea su par; intercambian información de alcance.
    - Alcance de vecinos: un router periódicamente prueba si el vecino es aún alcanzable; intercambia mensajes HELLO/ACK.
    - Actualización de rutas: pares periódicamente intercambian sus tablas de ruteo (vectores de distancia).
- EGP 1: BGP-4 (Border Gateway Protocol)

- El intercambio de información de rutas es entre cierto router de cada AS.
- No se preocupa por optimizar rutas, sino por determinar cómo alcanzar nodos.
- Cada AS tiene:
  - Uno o más routers de borde.
  - Un portavoz BGP que publica:
    - Redes locales.
    - Otras redes alcanzables.
    - Información de rutas.

## Nivel de red: otros protocolos

- **ARP (Address Resolution Protocol)**
  - Cuando un paquete es ruteado hasta llegar a la LAN en la que se encuentra el host destino, debemos determinar su dirección MAC para poder hacerle llegar el paquete. Sólo con la dirección MAC puede construir el paquete de nivel de enlace (por ejemplo, Ethernet) que contenga el mensaje.
  - Más aún, en cada hop del forwardo de un paquete, un router debe determinar la dirección MAC del siguiente router en el camino.
  - ARP es un protocolo que traduce direcciones IP en direcciones MAC.
  - Cada nodo en una LAN tiene una tabla ARP, que contiene dichos mapeos IP -> MAC. Inicialmente la tabla está vacía.
  - Cuando un nodo quiere resolver una dirección IP, si la MAC asociada no está en la tabla, emite un paquete ARP (encapsulado en un paquete de capa de enlace) con la consulta de la dirección IP.
    - Como el nodo no sabe cuál es la dirección MAC, la dirección de destino es broadcast. En Ethernet, esto es FF:FF:FF:FF:FF:FF.
    - Incluye su dirección IP y MAC. Esto le permite al nodo receptor agregar la entrada IP-MAC a su tabla ARP. El resto de los nodos no agregan la información IP-MAC del emisor a su tabla ARP pues probablemente no les sirva de nada.

## ARP Packet Format



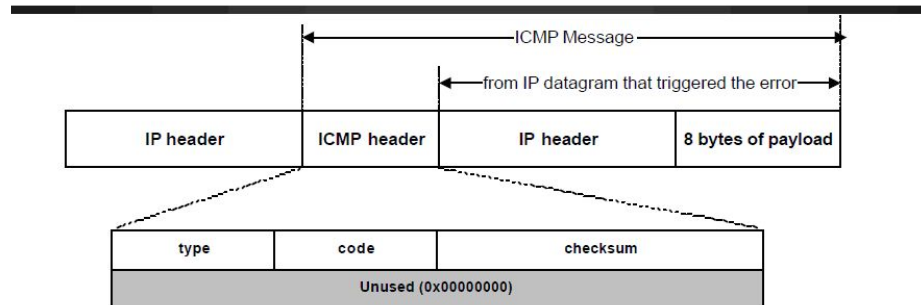
- El campo Operation code tendrá valor 1, lo que significa que es una consulta. Esta operación se llama *who has*.
- Cuando el nodo destino reconoce el paquete ARP con su IP, contesta con otro paquete ARP que contiene su MAC. Lo envía con Operation code de valor 2, que significa respuesta. Esta operación se llama *is at*.

- **DHCP (Dynamic Host Configuration Protocol)**

- Protocolo utilizado para asignar una configuración de red a un nodo que se acaba de conectar a una red.
- Cada interfaz de cada dispositivo ya trae una MAC de fábrica, así que eso no hay que configurarlo. Hay que asignarle una IP.
- Esta configuración se puede hacer a mano, pero esto tiene las desventajas típicas de un procedimiento que no es automático.
- En general, hay un nodo en la red, llamado **servidor DHCP** que se encarga de comunicarle la configuración a un nodo que se conecta.
- Cuando un nodo se conecta, emite un mensaje DHCP para descubrir el servidor DHCP (el mensaje es broadcast, dado que no sabemos su dirección IP). Cuando éste reconoce el mensaje, le responde al nodo con la información pertinente.
- No necesariamente habrá un servidor DHCP en cada red. En ese caso, habrá un *relay agent* que se encargará de recibir esos paquetes y reenviarlos al DHCP server.

- Los mensajes DHCP se envían utilizando UDP, pues provee multiplexación fácilmente.
- **ICMP (Internet Control Message Protocol)**
  - Protocolo para envío de mensajes de error y control.

## ICMP Error message

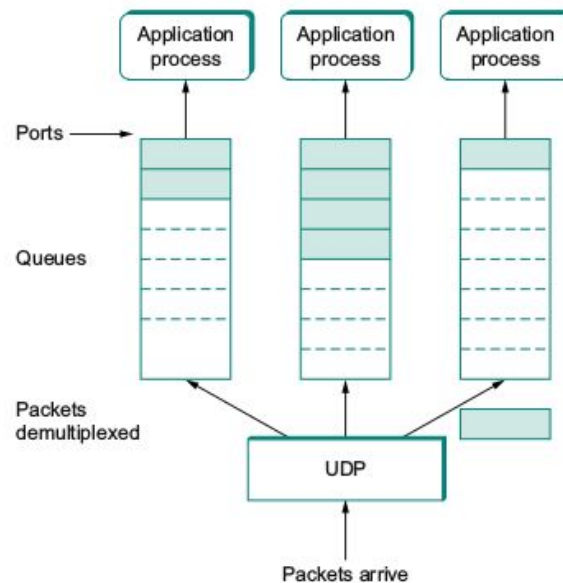


- ICMP error messages include the complete IP header and the first 8 bytes of the payload (typically: UDP, TCP)
- Utilizados para dar respuesta a distintos eventos que ocurren durante la transmisión de un paquete IP. Por ejemplo: el TTL llegó a 0, el host destino no responde, la verificación de checksum falló, etc.
- Dependiendo del valor del campo type es la función del mensaje. Algunos de ellos son:
  - 8: Echo Request. Solicita al receptor que conteste con otro mensaje ICMP. Usado para hacer **ping**.
  - 0: Echo Reply. Respuesta al Echo Request.
  - 3: Destination Unreachable.
  - 11: Time Exceeded. Respuesta en caso de TTL = 0. Usado para hacer **traceroute**.

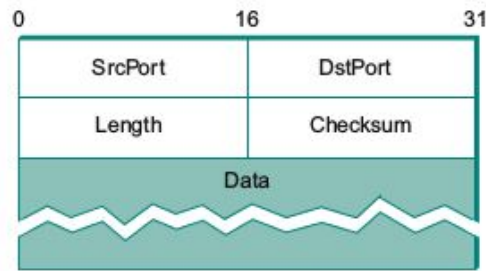
## Nivel de transporte

- El nivel de red permite la comunicación entre dos hosts cualquiera en Internet.
- Recordemos que el servicio de comunicación que brinda esa capa es best effort: descarta mensajes, los reordena, puede entregar múltiples copias, puede entregarlos luego de un tiempo arbitrariamente grande, etc.
- Queremos poder enviar datos a través de la red, en forma confiable.
- Queremos, además, que la comunicación tenga control de flujo (no sobrecargue al receptor) y control de congestión (no sobrecargue la red).

- Como en cada host puede haber varios procesos simultáneos que necesiten comunicarse por la red, necesitamos tener esta capacidad de administrar comunicaciones por proceso (hasta ahora sólo sabemos mandar paquetes de un host a otro, y no podemos saber a qué proceso pertenecen).
- Por lo tanto, el nivel de transporte es una capa montada sobre la de red, que busca asegurar:
  - Comunicación confiable.
  - Control de flujo y congestión.
  - Soportar múltiples procesos a nivel aplicación.
- **UDP (User Datagram Protocol)**
  - Es un protocolo de nivel de transporte, que busca únicamente soportar comunicación de proceso a proceso a través de la red.
  - Para identificar procesos en un host se utilizan **puertos**.
  - Un puerto es simplemente un identificador lógico (no confundir con puertos físicos o interfaces). Un proceso envía un mensaje a un puerto en una IP, y el proceso en el host con esa IP levanta mensajes de ese puerto.

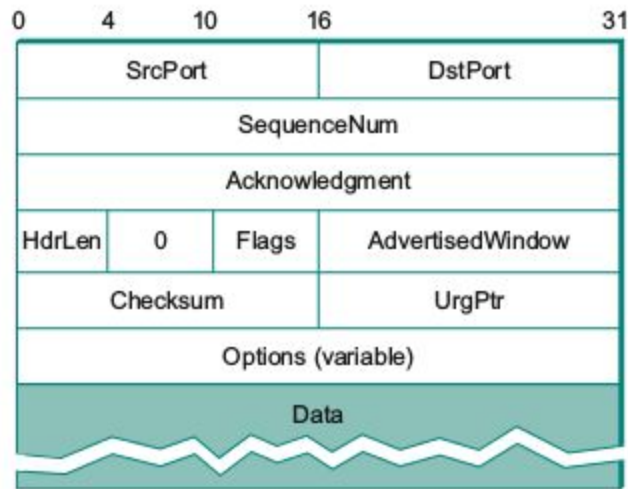


- Por lo tanto, cada proceso se identifica con una tupla (IP, puerto).
- Paquete UDP:



- Cuando un proceso quiere enviar un mensaje a otro, crea un paquete UDP con su puerto y el puerto destino, y luego lo encapsula en un paquete IP, cargado con su IP y la IP destino.
- ¿Cómo hace un proceso para saber el puerto del proceso destino?
  - El mecanismo más común es que servicios tipos (acceso a páginas web, descarga de mail, etc.) corran universalmente sobre los mismos puertos (por ejemplo, los servidores web HTTP escuchan el puerto 80).
  - Muchas veces, la comunicación se inicia en un puerto dado, en el que se están constantemente escuchando conexiones entrantes, y luego se la mueve a otro. En otras palabras, en el puerto fijo siempre hay un proceso escuchando conexiones, y cuando llega una se deriva a otro proceso en otro puerto.
  - A veces se utiliza un **port mapper**. Se pone un proceso sobre un puerto dado que escucha absolutamente todas las conexiones entrantes (no importa el servicio requerido). Al llegar un cliente, le indica el servicio con el que quiere comunicarse, y el port mapper le indica a qué puerto debe hablar.
- **TCP** (Transmission Control Protocol)
  - TCP provee todo lo que buscamos.
  - Es orientado a conexión.
  - Los datos se secuencian de a bytes, i. e. cada byte tiene un numerito. TCP brinda un servicio de byte stream.
  - Full duplex. Permite leer y escribir bytes.
  - Garantiza:
    - Transferencia de datos en orden
      - Esto es efectuado por el receptor a través de los números de secuencia de los bytes en los segmentos (un segmento es un paquete TCP).
    - Transmisión de datos libre de errores
    - Control de flujo

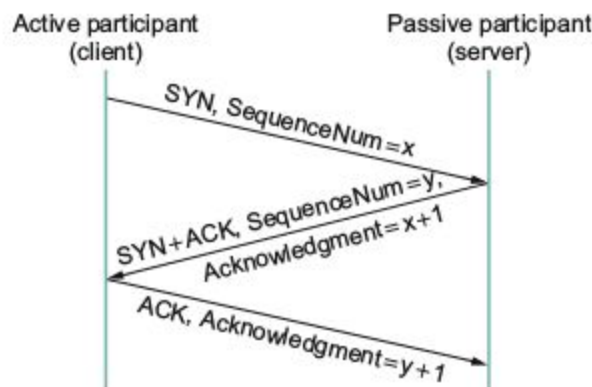
- Determina un límite al volumen de datos que el emisor puede enviar.
- El receptor informa cuántos bytes está dispuesto a aceptar a través de un campo en el header (similar a la RWS de sliding window de capa de enlace).
- Retransmisión de segmentos
  - Todo byte no reconocido es eventualmente retransmitido.
- Control de congestión
- Paquete TCP:



- Al igual que UDP, TCP utiliza puertos para multiplexar mensajes entre procesos.
- SequenceNum: número de secuencia de los bytes. Es el número de secuencia del primer byte contenido en ese segmento.
- Acknowledgment: el número de secuencia del siguiente byte que espera el emisor.
- Flags: bits de control. Los bits posibles son:
  - SYN y FIN. Para indicar inicio y fin de la comunicación.
  - RESET. El emisor no sabe qué hacer a partir de un segmento TCP recibido (está confundido), y desea abortar la conexión.
  - PUSH. Indica al emisor que debe entregar los datos a la aplicación inmediatamente después del arribo del segmento. No esperar a que el buffer total haya sido recibido.
  - URG. Indicando que el segmento contiene cierta porción de información urgente.
  - ACK. Indica que se están reconociendo bytes recibidos. Los ACK son acumulativos.

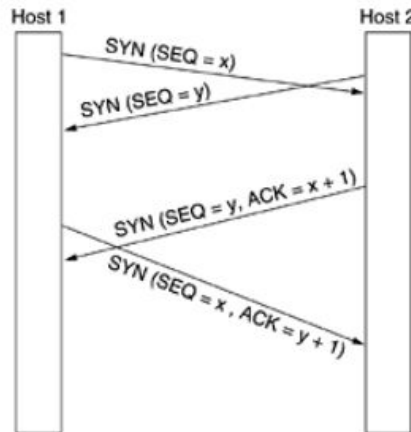


- Options: para poder agregar extensiones, no cubiertas por el header regular.
- Checksum: pseudo header (campos de IP) + TCP header + data.
- El protocolo distingue tres fases de operación:
  - *Establecimiento de la conexión*, en donde ambos extremos sincronizan números de secuencia y reservan estado.
  - *Transferencia de datos*, que es la etapa en la que los extremos se intercambian información.
  - *Cierre de conexión*, donde se cierra el canal y liberan recursos.
- Apertura de conexión
  - El algoritmo usado por TCP para establecer una conexión se llama **three-way handshake**.
    - A envía a B un paquete con SYN, indicando que quiere abrir una conexión.
    - B reconoce el SYN, y le envía un ACK más un SYN, indicando que él también quiere abrir una conexión con A.
    - A reconoce el SYN de B con un ACK.

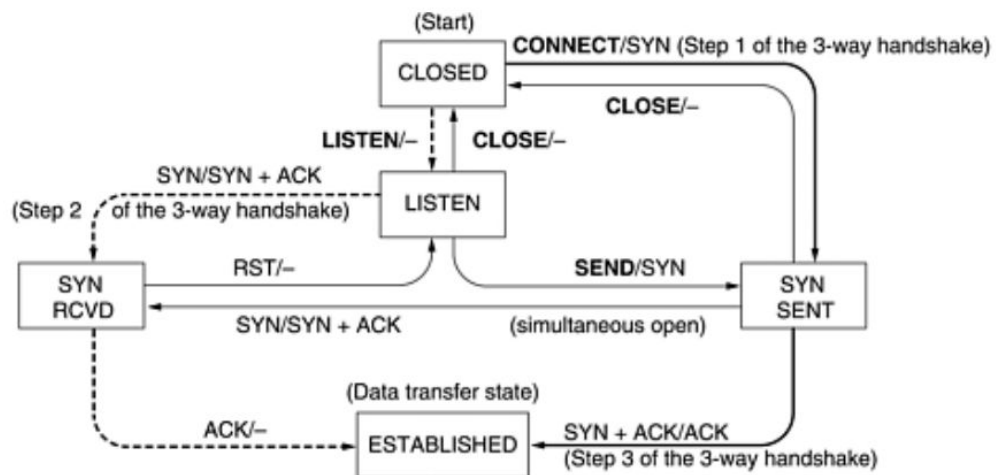


- El cliente es un actor activo (es el que busca iniciar la conexión) y el servidor es pasivo (simplemente escucha conexiones entrantes).
- Observaciones:
  - Los SYN se secuencian, y cuentan como 1 byte. Esto es porque hay que reconocerlos con ACK, y sólo se puede hacer ACK de cosas secuenciadas.
  - Los SequenceNum iniciales no son 0 sino un número al azar. La finalidad de esta elección es evitar reencarnaciones de segmentos de conexiones anteriores. Es claramente más difícil que justo un frame que quedó navegando de una conexión vieja pueda malinterpretarse como actual, comenzando con un número de secuencia aleatorio.

- Puede ocurrir (aunque no es usual) que ambos extremos intenten iniciar una conexión entre sí, al mismo tiempo. Esto se denomina **apertura simultánea**, y es contemplado por TCP.

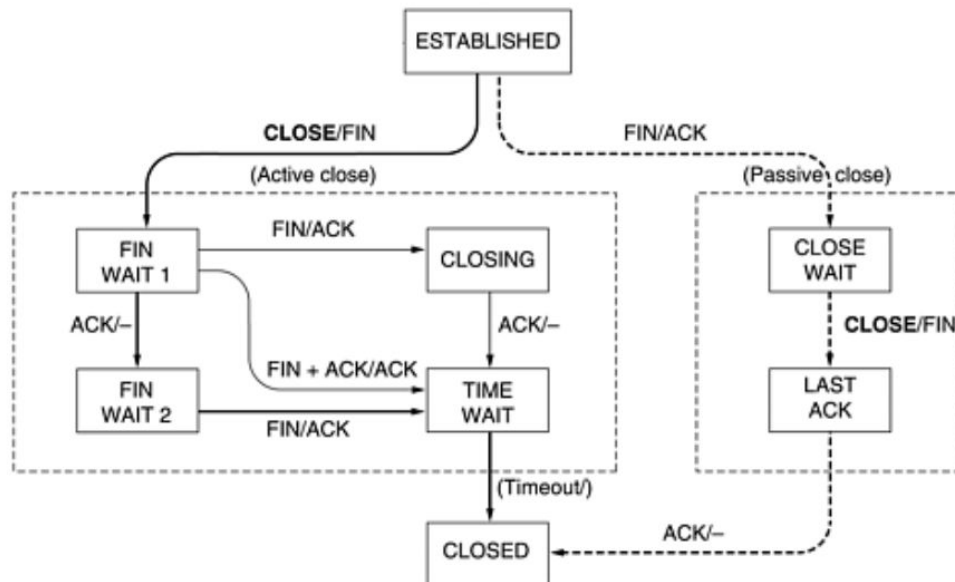
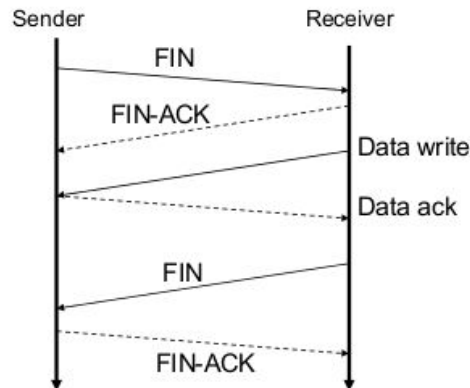


- Notar que cuando cada uno recibe el SYN del otro, responde con un SYN + ACK, y el SYN que envía tiene el mismo número de secuencia que el que envió inicialmente (es decir, reafirma el SYN inicial).
- El siguiente diagrama de estados describe la apertura de conexión TCP:



- Hay transiciones invisibles en el diagrama, correspondientes a retransmisiones de un mensaje. En caso de sucesivas retransmisiones fallidas, TCP vuelve al estado CLOSED.
- Cierre de conexión
  - El cierre de conexión debe ser ejecutado por ambos extremos. Cada conexión se cierra en una dirección.

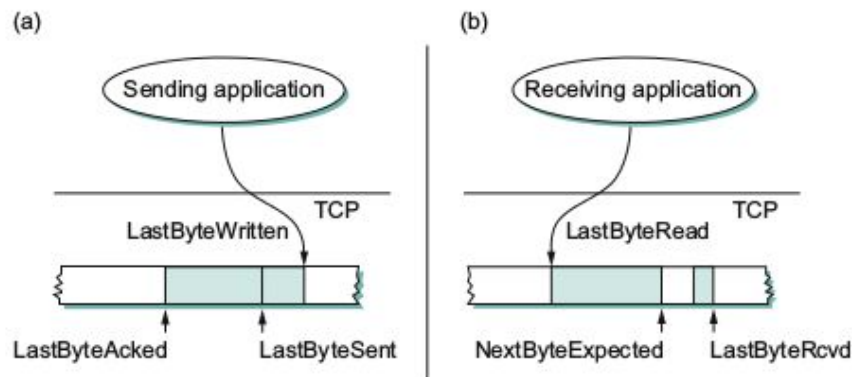
- Podría ocurrir que un extremo A cierre la conexión, pero que el otro extremo B la mantenga abierta. Esto implica que A ya no puede enviar datos, pero B sí puede (y por ende, A sigue pudiendo recibir datos).
- Por ende, hay tres posibilidades: A cierra primero que B, B cierra primero que A, y ambos cierran al mismo tiempo.
- El cierre típico es el llamado **four-way handshake**.
  - A envía a B un paquete con FIN, indicando que no enviará más datos.
  - B reconoce este FIN enviando un ACK.
  - (B sigue enviándole datos a A.)
  - B envía a A un paquete con FIN, para cerrar su extremo de la conexión.
  - A reconoce el FIN enviando un ACK.



- También existe la posibilidad de **cierre simultáneo**. Esto ocurre si ambos mandan el FIN al mismo tiempo.
- Notar que al llegar a TIME WAIT, necesariamente se espera durante un rato hasta que ocurra un timeout y sólo allí se pasa a estado CLOSED. Esto tiene dos causas bien específicas:
  - La primera es esperar a que todo paquete de la conexión que pudiera estar vivo dando vueltas por Internet, muera. Si bien al llegar a TIME WAIT tenemos la garantía de que todos los datos han sido transmitidos exitosamente, puede que haya segmentos duplicados viajando por ahí. Si se abriera una nueva conexión entre las partes, esos segmentos viejos podrían interpretarse como segmentos válidos de la nueva conexión. Se suele esperar 120s que es lo máximo que se estipula que puede vivir un paquete dando vueltas por Internet.
  - La segunda es que si un host A pasa de TIME WAIT a CLOSED sin esperar, y al final resulta que el otro extremo de la conexión B no recibió el último ACK, B retransmitirá su último mensaje, al que A, en estado CLOSED, responderá con RESET (no sabe qué hacer). B cree que A sigue conectado y está pidiendo abortar la conexión.
- Sliding Window en TCP
  - El protocolo de Sliding Window de TCP busca asegurar dos cosas:
    - Transmisión confiable y ordenada. Es decir, que los bytes se lean en orden, que no lleguen duplicados y que lleguen todos. En este sentido, el protocolo es similar a Sliding Window de capa de enlace.
    - Control de congestión. Un segmento TCP viaja a través de varios routers y enlaces. Como esos enlaces también son usados por otras comunicaciones, podrían congestionarse. Esto es percibido por cada participante, cuando el delay aumenta o cuando los paquetes empiezan a perderse (pues son descartados por los routers intermedios) o llegan fuera de orden. TCP reconoce esta situación y actúa para detener el congestionamiento de la red.
    - Control de flujo. En Sliding Window de capa de enlace teníamos control de flujo, a través de las ventanas de emisión y recepción. ¿Por qué acá no podemos usar ese mismo esquema? Porque dependiendo de la intensidad de trabajo que esté realizando cada participante, su velocidad de procesamiento y recursos disponibles podría variar. Por ejemplo, si la ventana es muy grande, el emisor

podría inundar de datos al receptor, que quizás no puede absorber. El receptor debería poder avisar de esto al emisor, para que éste achique su ventana de emisión.

- Cada participante en la comunicación tiene un buffer, en el cual va a escribiendo (en el caso del emisor) o leyendo (en el caso del receptor) bytes. Es una situación análoga a la de Sliding Window en capa de enlace, solo que en lugar de frames tenemos bytes.
- Transmisión confiable y ordenada
  - Cada participante mantiene una serie de variables, que representan punteros a bytes (números de secuencia).



- Invariantes
  - Emisor:  
 **$\text{LastByteAcked} \leq \text{LastByteSent} \leq \text{LastByteWritten}$**
  - Receptor:  
 **$\text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteRcvd} + 1$**
- Observar que en este modelo, a diferencia de Sliding Window en capa de enlace, tenemos las variables LastByteWritten y LastByteRead, que ponen en juego a las velocidades de lectura y escritura (por ejemplo, si el procesamiento de los datos recibidos es lento, la velocidad de lectura será lenta).
- Control de flujo
  - Llamemos MaxRcvBuffer y MaxSendBuffer al tamaño del buffer del receptor y emisor, respectivamente.
  - Receptor:
    - Los bytes que recibió pero que no ha leído aún, deben entrar en el buffer:  
 **$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$**
    - Debe indicarle al emisor cuánto es la máxima cantidad de bytes que puede recibir sin necesidad de droppear datos.

Llamamos AdvertisedWindow a esta cantidad, y es el tamaño de ventana que el receptor quiere que el emisor adopte:

**AdvertisedWindow = MaxRcvBuffer - ((NextByteExpected - 1) - LastByteRead)**

Notar que la AdvertisedWindow es un campo del paquete TCP y se transmite haciendo piggybacking junto con el resto de la información.

■ Emisor:

- Todos los bytes no reconocidos por el receptor, deben entrar en el buffer:

**LastByteWritten - LastByteAcked <= MaxSendBuffer**

- Debe asegurarse de no transmitir más datos que la AdvertisedWindow:

**LastByteSent - LastByteAcked <= AdvertisedWindow**

- Los datos enviados pero que aún no fueron reconocidos se denominan **datos en vuelo**. Esta cantidad es

**FlightSize = LastByteSent - LastByteAcked**

- La ventana efectiva es la cantidad de datos que en este momento, teniendo en cuenta los datos en vuelo y la última advertised window, podemos enviar:

**EffectiveWindow = AdvertisedWindow - FlightSize**

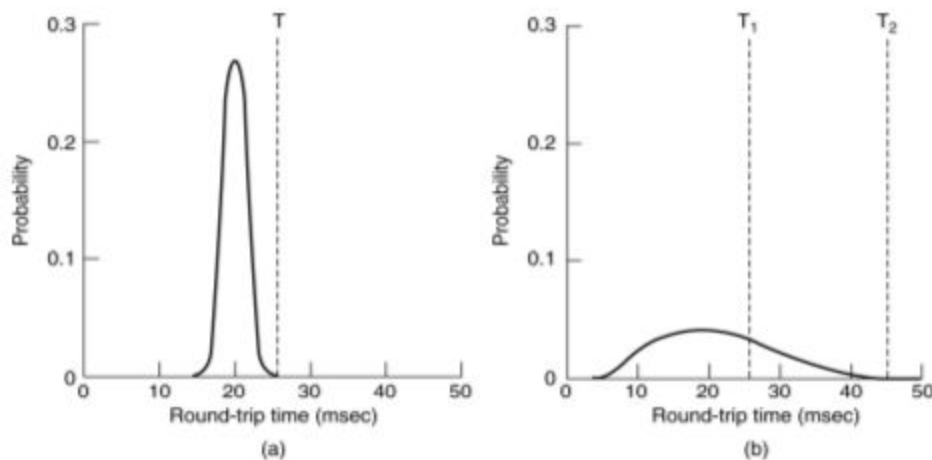
- Observación: si la advertised window llega a 0, el emisor debe dejar de enviar datos. Entonces, el receptor también dejará de emitir ACKs, y no hay forma de que pueda cambiar su advertised window. Lo que se hace en este caso es que el emisor periódicamente manda un segmento de 1B para producir una respuesta del otro extremo y ver si es posible aumentar la ventana.

■ Números de secuencia

- El frame TCP provee 32bits para secuenciar bytes. Esto parece mucho, pero en conexiones rápidas puede agotarse rápido, haciendo que los números de secuencia se reinicien peligrosamente rápido.

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

- Recordemos que se considera que un paquete puede vivir hasta 120s dando vueltas por la red. Entonces, en una conexión rápida, que agota todos los números de secuencia más rápido que eso, no se puede garantizar que no haya reencarnaciones.
- Retransmisión y timeout
  - Como ya venimos diciendo, el RTT en una transmisión a través de la red es altamente variable.



- (a) Función densidad de probabilidad para tiempos de llegadas de ACK a nivel 2
- (b) Función densidad de probabilidad para tiempos de llegadas de ACK TCP

- Por ende, el RTO (Retransmission TimeOut) debe ir adaptándose a lo largo de la conexión.

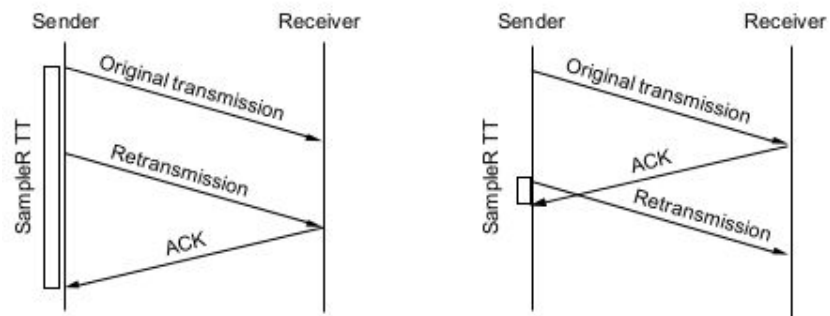
- **Retransmisión Adaptativa**

- Algoritmo original

- Cada vez que recibe una respuesta, toma ese RTT como muestra, y recalcula el RTT estimado:

$$\text{EstimatedRTT} = a * \text{EstimatedRTT} + (1 - a) * \text{SampleRTT}$$

- El número  $\alpha$  es un real entre 0 y 1. A mayor  $\alpha$ , más importancia le damos a la estimación histórica del RTT. A menor  $\alpha$  (mayor  $1 - \alpha$ ) más importancia le damos al sample tomado.
- Se suele tomar  $\alpha$  entre 0.8 y 0.9.
- Define  $RTO = 2 * EstimatedRTT$
- Problema: el ACK reconoce datos, pero no indica a cuál de las transmisiones (podríamos haber retransmitido los mismos datos varias veces) corresponde.



#### ■ Algoritmo Karn/Partridge

- Estima el RTT igual que antes, pero cuando hay retransmisiones, no toma mediciones del RTT. Ante cada retransmisión duplica el RTO (es como un exponential backoff). La idea de duplicar el RTO es aflojar con las retransmisiones para no seguir sobrecargando la red.

#### ■ Algoritmo de Jacobson/Karels

- Utiliza una especie de desviación para ir retocando la estimación.

#### ■ RFC 6298

- Hace una cuenta manteniendo dos variables: SRTT (Smoothed RTT) y RTTVAR (RTT Variation).

- Transmisión eficiente
  - Queremos maximizar el uso de un segmento. No queremos mandar muchos segmentos con pocos datos. No queremos mandar muchos ACKs de pocos bytes.
  - MSS (Maximum Segment Size)
    - Es el máximo tamaño que puede tener un segmento TCP. Es aproximadamente la máxima cantidad de payload que podemos meter en un segmento.



- Se suele setear al tamaño del mayor segmento TCP que no causa fragmentación del paquete IP en el enlace local (es decir, el MTU de nuestra red local).
- Lo ideal es enviar segmentos llenos, i. e. de tamaño MSS.
- ¿Cuándo debe el emisor enviar un segmento? **Algoritmo de Nagle.**

**if (el tamaño de la advertised window y los datos disponibles en el buffer  $\geq$  MSS) {**

**enviar un segmento full;**

**} else {**

**if (hay datos en vuelo no reconocidos) {**

**bufferear el nuevo dato hasta que llegue un ACK;**

**} else {**

**enviar todos los datos ahora;**

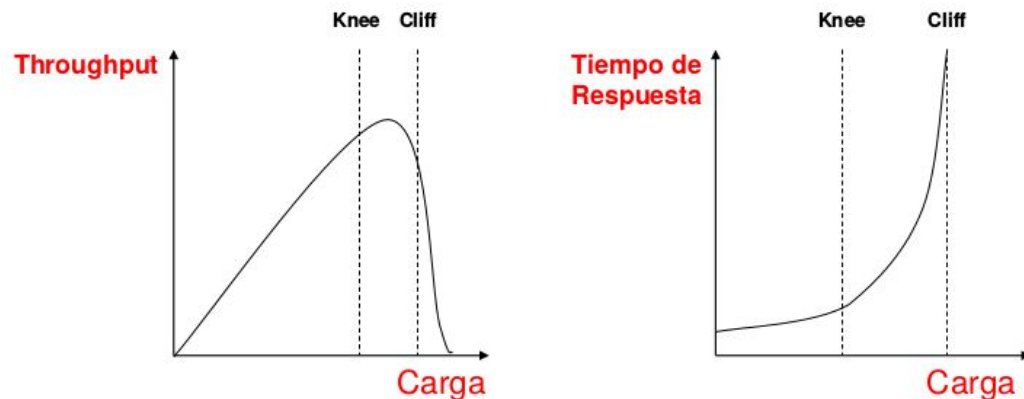
**}**

**}**

- Idea: si ya tengo suficientes datos, enviarlos. Si no, esperar un ACK del otro extremo (es decir, usar el ACK como un timer).
- **Síndrome de la ventana estúpida**
  - El receptor pone una advertised window muy chiquita, porque se acaba de abrir después de haberse cerrado. Esto significa que el receptor estaba leyendo lento, y el emisor lo llenó de datos.
  - El emisor transmite unos pocos bytes, llenando esa ventana.
  - Como receptor lee lento, no alcanza a ampliar la advertised window, que se mantiene pequeña.
  - El emisor vuelve a transmitir unos pocos bytes. Así sucesivamente.
  - Esto hace que todos los segmentos TCP tengan muy poco payload.
  - **Solución de Clark**
    - Cuando se anuncia una advertised window nula, el receptor debe esperar a tener espacio MSS, o la mitad de su buffer disponible, antes de informar un cambio en la advertised window.
- ¿Cuándo el receptor debe enviar ACK?
  - Juntar un bloque de datos recibidos antes de mandar un ACK.
  - Si estamos recibiendo segmentitos chicos, no tiene sentido emitir un ACK por cada uno de ellos.

# Congestión

- Congestión: la red se satura y se empiezan a perder paquetes.
  - Causa: las queues de los routers están saturadas y descartan paquetes.
  - Si los usuarios de la red no paran de mandar datos, las colas eventualmente no tienen espacio suficiente y empiezan a droppear paquetes. Los protocolos corriendo en los hosts timeoutean, provocando el reenvío de paquetes. La red entra en un estado de congestión del cual no puede salir si no se para la pelota.



- Soluciones:
  - Congestion control
  - Congestion avoidance.
- Criterios para medir congestión:
  - $\text{Power} = \text{Throughput} / \text{Delay}$ 
    - Empieza a haber congestión cuando la carga es tal que el throughput empieza a caer, y el delay empieza a subir.
  - Fairness
    - Todos los hosts reciben una porción equitativa de la capacidad de la red.
- Control de congestión
  - Dos formas de *controlar* la congestión:
    - En los routers: haciendo queueing inteligente.
    - En los extremos de la conexión: con control de congestión de TCP.
  - **Queueing**
    - Los routers tienen una (o varias) queues (buffers) en las que meten los paquetes que van despachando.
    - **FIFO**. Todos los paquetes (independientemente de los extremos de la comunicación) van a parar al mismo buffer.

- Problema: un host podría fácilmente saturar un router. No hay control sobre qué porción de la queue se le asigna a cada host.
- **FQ (Fair Queueing)**. Cada flujo (comunicación entre dos puntas) tiene asignado una cola de cierto tamaño. Si la llena, sólo se droppean sus paquetes excedentes. Se despachan los paquetes de cada cola de modo tal de asegurar fairness (no es tan simple como hacer round robin entre las colas, porque podría haber una cola con paquetes mucho más grandes que otra; hay que tener en cuenta el tamaño de los paquetes).
- **TCP Congestion Control**
  - Se hacen agregado al sistema de sliding window de TCP.
    - Se define una nueva variable **CongestionWindow**, que representa cuántos datos la red está disponible a aceptar antes de congestionarse (es análogo a la AdvertisedWindow, pero en lugar de decir cuántos datos se bancaría el receptor, dice cuántos se bancaría la red).
    - Ahora el emisor tiene que enviar datos teniendo en cuenta no sólo sin pasarse de la AdvertisedWindow, sino también de la CongestionWindow. Las cuentas cambian sutilmente:
 
$$\text{MaxWindow} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$

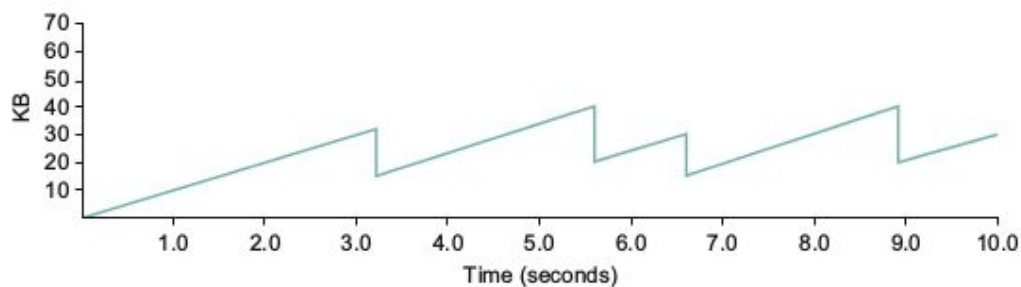
$$\text{EffectiveWindow} = \text{MaxWindow} - \text{FlightSize}$$
  - ¿Cómo hace TCP para determinar CongestionWindow?
    - Idea: TCP va tanteando cuánto puede mandar sin que *haya congestión*.
    - TCP asume que hay congestión cuando timeoutea un paquete (es decir, un byte no es reconocido luego de un RTO), ya que esto es posiblemente porque el paquete fue descartado en algún router intermedio).
    - Comenzamos con una CongestionWindow de MSS. Va incrementándola hasta que eventualmente ocurre un timeout. En ese momento, la disminuye.
    - Incrementamos la CongestionWindow en cada ACK de nuevos datos. Cada ACK nuevo es tomado como un signo de mayor capacidad disponible de la red.
  - **Additive Increase/Multiplicative Decrease (AIMD)**

- La ventana de congestión se incrementa de a poco (sumando), y se decrementa abruptamente (dividiendo por 2).
- La lógica de esto es que queremos salir de un estado de congestión lo más rápido posible.
- Incrementos: la idea es incrementar de a un segmento (MSS bytes) por RTT (por ráfaga de envíos; que podemos pensar que en cada RTT hay una ráfaga). Como recibimos muchos ACKs por RTT (hasta  $\text{CongestionWindow} / \text{MSS}$ , uno por cada segmento enviado), cada vez que recibimos un ACK incrementamos un poquito:

$$\text{Increment} = \text{MSS} * (\text{MSS} / \text{CongestionWindow})$$

$$\text{CongestionWindow} += \text{Increment}$$

- Decrementos:  
 $\text{CongestionWindow} /= 2$



#### ■ Slow Start

- AIMD sirve cuando estamos cerca de llegar al punto de congestión. Por el contrario, si arrancamos con una ventana de congestión chica y la aumentamos lentamente, probablemente tardemos mucho en llegar al punto de congestión, y desperdiciemos capacidad de la red.
- Slow Start consiste en duplicar la CongestionWindow en cada RTT. Más específicamente, aumentar la ventana en un MSS por RTT.
- Slow Start se usa en dos momentos.
- *Al principio de la conexión*, para acelerar la convergencia al punto de congestión. Cuando ocurre timeout, se divide la CongestionWindow en 2 y comienza AIMD.
- *Cuando ocurre un timeout* en el medio de la conexión, es porque hay un paquete intermedio que se perdió. Durante ese tiempo, el receptor mueve su LastByteRead (lee datos en su buffer) pero no mueve LastByteAcked. Eventualmente

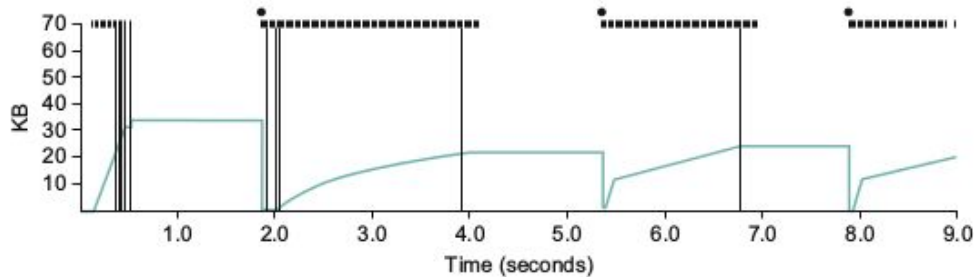
eso provoca que la AdvertisedWindow esté completamente abierta. El emisor podría recibir esta señal y decidir retransmitir a partir del timeout, con una ventana de tamaño AdvertisedWindow. Lógicamente un aumento de la ventana tan abrupto podría congestionar la red. Por esta razón, cuando hay un timeout, la retransmisión es con slow start.

- En este caso se pone

$$\text{CongestionThreshold} = \text{CongestionWindow} / 2$$

$$\text{CongestionWindow} = \text{MSS}$$

- Se hace slow start hasta que la ventana de congestión tenga tamaño CongestionThreshold. A partir de ahí se hace AIMD.



■ **FIGURE 6.11** Behavior of TCP congestion control. Colored line = value of CongestionWindow over time; solid bullets at top of graph = timeouts; hash marks at top of graph = time when each packet is transmitted; vertical bars = time when a packet that was eventually retransmitted was first transmitted.

## ■ Fast Retransmit

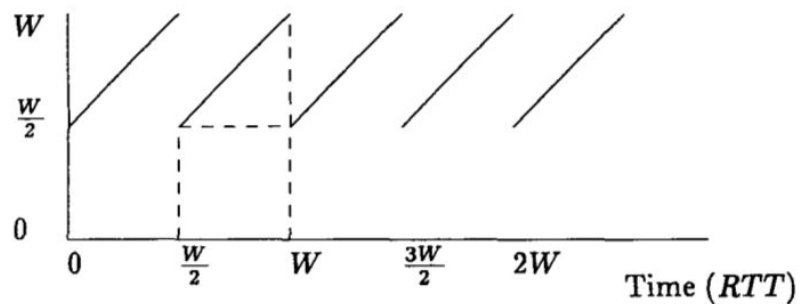
- Observar en el grafiquito anterior, que hasta que ocurre un timeout hay una gran meseta, que es tiempo muerto.
- Fast Retransmit consiste en no esperar al timeout para determinar que hubo congestión.
- Necesitamos que el receptor emita un ACK por cada segmento recibido, aún si es un segmento fuera de orden. Es decir, el receptor emite ACKs duplicados (ACKs de un mismo número de secuencia).
- Cuando el emisor recibe 3 ACKs duplicados de un mismo byte, dispara la retransmisión y asume que hubo congestión, poniendo  $\text{CongestionThreshold} = \text{CongestionWindow} / 2$  y  $\text{CongestionWindow} = \text{MSS}$ , y hace Slow Start.

## ■ Fast Recovery

- Cuando se hace Fast Retransmit, en lugar de poner  $\text{CongestionWindow} = \text{MSS}$  y hacer Slow Start hasta  $\text{CongestionThreshold}$ , poner directamente  $\text{CongestionWindow} /= 2$  y hacer AIMD. Sólo hacer Slow Start si efectivamente ocurre un timeout.

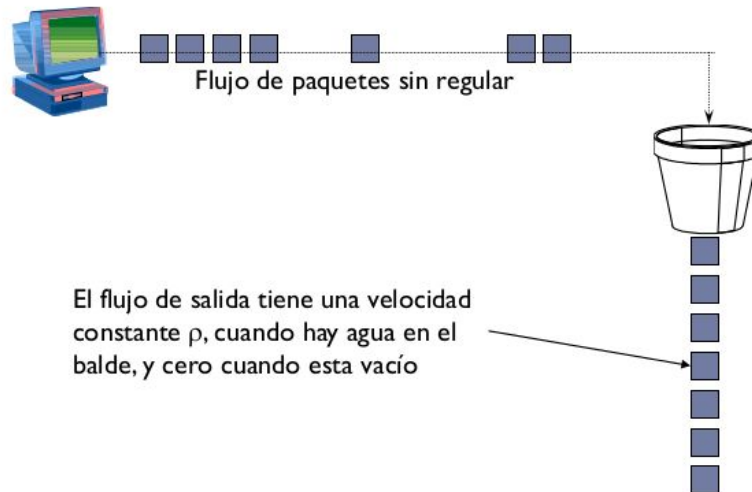
○ **Límite del throughput TCP**

- En el paper de Mathis hacen una cuenta de la estimación el throughput de TCP, bajo ciertas simplificaciones.
- Asumen un modelo que se congestiona exactamente con una ventana de  $W$  segmentos (es decir, el tamaño en bytes sería  $W * \text{MSS}$ ).
- Además asume que la ventana se va a  $W/2$  en un timeout, y a partir de allí se hace AIMD.
- Se asume que apenas se llega a una ventana de tamaño  $W$ , se produce un timeout (el tiempo de timeout es despreciable).



- (El eje vertical se mide en cantidad de segmentos.)
- (El eje horizontal se mide en RTTs; i. e.  $0 \text{ RTT}$ ,  $(W/2) \text{ RTT}$ , etc.)
- (Recordar que por cada RTT se aumenta la ventana de congestión en un segmento i. e.  $\text{MSS}$  bytes. Por eso es que la ventana crece linealmente.)
- Asumen que la probabilidad de perder un paquete es  $p$ . Por ende, aproximadamente cada  $1/p$  paquetes manda uno que se pierde.
- Lo anterior implica que se mandan  $1/p$  paquetes durante cada ciclo (cada diente de sierra). Como la cantidad de paquetes que se envían es el área bajo la curva, resulta que  $W = \sqrt{8/(3p)}$ . Observar que esto pone el punto de congestión en función de  $p$ .
- Luego hacemos la cuenta  
Throughput = (segmentos enviados por ciclo) / (tiempo por ciclo)  
Y resulta que da **Throughput =  $C / (\text{RTT} * \sqrt{p})$**  para cierta constante  $C$ .

- En particular, la capacidad es  $BW = (MSS * C) / (RTT * \sqrt{p})$
  - **Explicit feedback vs. implicit feedback**
    - Implícito: la red descarta paquetes, lo cual produce un timeout o ACKs repetidos, lo cual es una señal para los extremos. Por ejemplo: TCP.
    - Explícito: los routers proveen información precisa a las fuentes.
    - Para implementar feedback explícito se requiere agregar lógica tanto en los routers como en los hosts.
    - Ejemplo: DECbit.
- Congestion Avoidance
  - **RED (Random Early Detection)**
    - Mecanismo de encolamiento que consiste en descartar un paquete cuando se observa una inminente congestión.
    - En otras palabras, descarta anticipadamente un paquete para que se tomen medidas en los extremos, y no llegar a la congestión.
    - Cada vez que ingresa un paquete:
      - Se calcula un AvgLen de la cola, utilizando un SampleLen, igual que como hacíamos con EstimatedRTT.
      - Si  $AvgLen \leq MinThreshold$ , encolar.
      - Si  $MinThreshold < AvgLen < MaxThreshold$ , descartar con probabilidad  $p$ .
      - Si  $MaxThreshold \leq AvgLen$  descartar paquete.
    - La probabilidad  $p$  se calcula en cada paso, y es más grande cuanto más cerca de  $MaxThreshold$  está  $AvgLen$ .
    - Problema: es imparcial con conexiones de bajo ancho de banda (baja velocidad). Las conexiones que más datos transmiten usarán una mayor parte de la queue.
  - **FRED (Flow Random Early Detection)**
    - Soluciona el problema de la imparcialidad, manteniendo umbrales e información de la ocupación del buffer para cada flujo (conexión).
- Políticas de tráfico
  - Mecanismos para controlar tráfico irregular (bursty), que impacta en la congestión.
  - **Traffic Sharping** es un mecanismo que trata de guiar la congestión, forzando a los paquetes a transmitirse a una velocidad más predecible.
  - Trata de mantener el tráfico constante.
  - Dos algoritmos de Traffic Sharping:
    - **Leaky Bucket**
      - Fuerza un patrón de tráfico de salida rígido.



#### ■ Token Bucket

- Permite picos de tráfico pero sólo por breves instantes de tiempo.

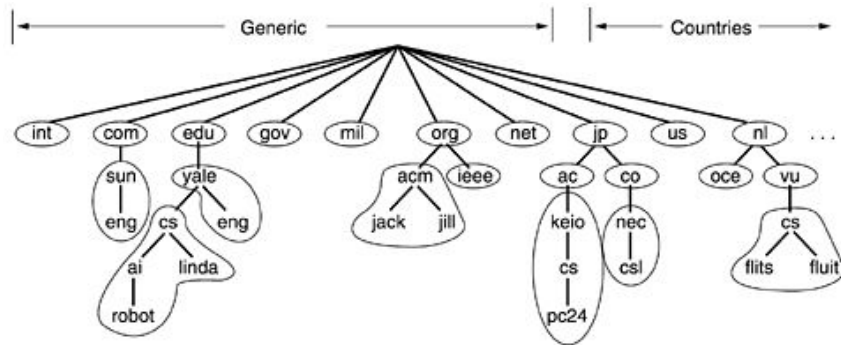
## Nivel de aplicación

### ● DNS (Domain Name System)

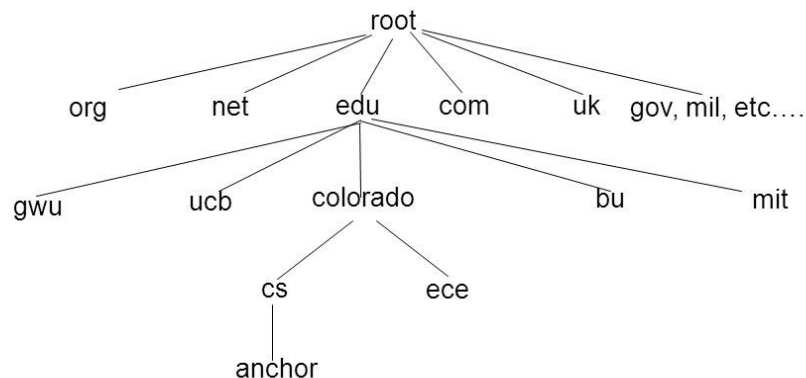
- Sistema de traducción de nombres de dominio (por ejemplo el nombre de una página web, o un email) en direcciones IP. En realidad puede traducir un nombre en cualquier tipo de valor.
- Esta traducción es necesaria no sólo por comodidad (es difícil recordar IPs), sino porque la IP asociada a un dominio podría cambiar a lo largo del tiempo. Inclusive, un dominio podría tener asociadas varias IPs.
- El protocolo es de tipo cliente-servidor. Un cliente pide la **resolución** de un nombre a un servidor DNS. Corre sobre UDP. El servidor escucha el puerto 53.
- Un dominio se descompone en partes, dando lugar a una jerarquía de dominios. Por ejemplo, *dc.uba.ar.* se descompone en *dc.*, *uba.* y *ar.* Esto significa que el dominio se encuentra abajo de *ar.*, y a su vez *dc.* se encuentra abajo de *uba.ar.* Esta es la base para hacer de DNS un sistema distribuido.
- **Zonas**



- Grupos de varios dominios. Son disjuntas.



- Cada zona tiene un name server responsable, llamado servidor primario. Un name server puede administrar varias zonas.
- También tiene varios servidores secundarios, que mantienen copias de la información del servidor primario, la cual actualizan periódicamente.
- Hay un servidor de nombres **root**. Éste conoce los servidores de nombres de las zonas del primer nivel, como .com, .edu, etc.



#### ○ Tipos de servidores

- **Autoritativo.** El servidor primario y los secundarios de una zona se llaman autoritativos, porque estos tienen la posta.
- **Local** (o Resolver DNS). Son servidores locales que dan respuesta a consultas DNS. En caso que no sepan resolver el dominio, efectúan consultas a servidores autoritativos.
- Los servidores locales cachean todas las respuestas que obtienen de servidores autoritativos. Esto les permite resolver algunas

consultas sin necesidad de recurrir nuevamente a servidores autoritativos.

- Si la información viene de un servidor autoritativo, está respaldada por el administrador de la zona. Si proviene de la cache de un servidor local, la información no es autoritativa y se puede recurrir a un servidor de la zona para obtener una respuesta confiable.
- Tabla DNS
  - Cada servidor DNS mantiene una tabla con registros DNS.

nombre de dominio	TTL	clase	tipo	valor	
www.dc.uba.ar.	600	IN	CNAME	www-1.dc.uba.ar.	ALIAS
www-1.dc.uba.ar.	600	IN	CNAME	dc.uba.ar.	
dc.uba.ar.	600	IN	A	157.92.27.21	Address IPv4
dc.uba.ar.	600	IN	NS	ns2.uba.ar.	
dc.uba.ar.	600	IN	NS	ns-1.dc.uba.ar.	Name Server
dc.uba.ar.	600	IN	NS	ns-2.dc.uba.ar.	
dc.uba.ar.	600	IN	NS	ns1.uba.ar.	
ns1.uba.ar.	1451	IN	A	157.92.1.1	
ns1.uba.ar.	6847	IN	AAAA	2001:1318:100c:1::1	Address IPv6
ns2.uba.ar.	1451	IN	A	157.92.4.1	
ns2.uba.ar.	6847	IN	AAAA	2001:1318:100c:4::1	
ns-2.dc.uba.ar.	600	IN	A	157.92.27.253	

- TTL es la vida restante del registro. Una vez que TTL llega a 0, esa entrada se vuelve inválida (hay que volver a consultar a un servidor autoritativo).
- Clase (en general es IN, de Internet).
- Tipos (algunos):
  - **SOA (Start Of Authority)**. Indica el nombre del servidor primario y brinda información sobre los updates.
  - **A**. Dirección IPv4.
  - **MX**. Nombre de un mail server.
  - **NS**. Nombre de un name server.
  - **CNAME**. Alias de un nombre. Sirve, por ejemplo, cuando el nombre del servidor en realidad es alguna cosa más fea que el nombre que el cliente usa.

## ■ Registros SOA

```
dc.uba.ar.      IN      SOA      ns1.dc.uba.ar.      admines.dc.uba.ar. (
2014052000
4h
1h
4w
2h
)
```

- Además de indicar el nombre del servidor primario, brindan información sobre los updates (los parámetros entre paréntesis): ¿cada cuánto actualizar la información? ¿cuánto tiempo esperar para hacer un retry de pedido de información al primario? Entre otras cosas.

## ○ Resolución de consulta DNS

- Un cliente efectúa una consulta DNS. Ésta tiene la forma (nombre de dominio, clase, tipo). Por ejemplo:

```
www.dc.uba.ar.      IN      A
```

- La consulta se envía al servidor DNS local. Supongamos que no conoce la dirección de ese dominio. Más aún, lo único que conoce es el servidor de nombres root. Le envía la consulta a ese servidor.
- De root cuelga la zona .ar, así que el servidor root tiene dos registros

```
ar.      IN      NS      c.dns.ar.
c.dns.ar.  IN      A      200.108.148.50
```

Le responde al DNS local con la información de c.dns.ar.

- El DNS local envía la consulta a c.dns.ar. Este servidor puede que no sepa resolver www.dc.uba.ar. pero al menos debe tener información sobre uba.ar. que cuelga de su zona. Por ejemplo:

```
uba.ar.      IN      NS      ns1.uba.ar.
ns1.uba.ar.  IN      A      157.92.1.1
```

Le devuelve al servidor local la información sobre ns1.uba.ar.

- El servidor local le manda la consulta al name server ns1.uba.ar. Este servidor posiblemente no sepa resolver www.dc.uba.ar, aunque sí sabe la dirección del name server de dc.uba.ar.

```
dc.uba.ar.      IN      NS      ns1.fcen.uba.ar.
ns1.fcen.uba.ar.  IN      A      157.92.32.4
```

Le responde la información de ns1.fcen.uba.ar. al servidor local.

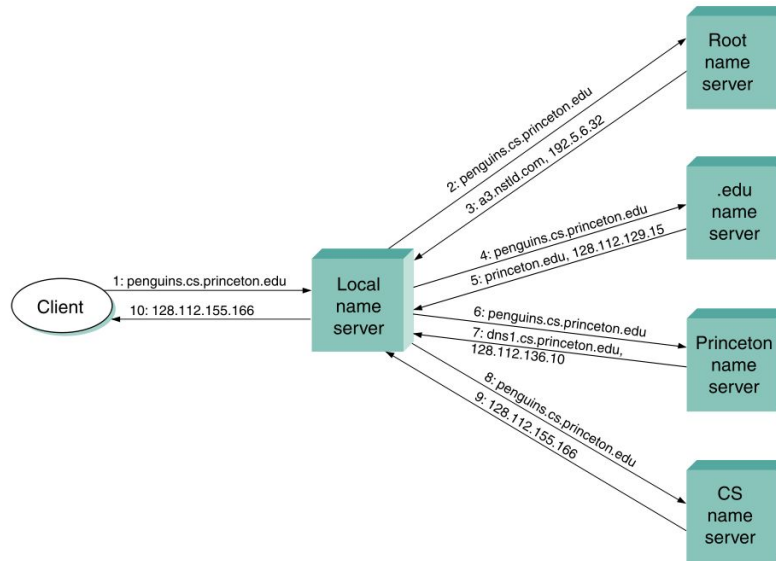
- Finalmente, ns1.fcen.uba.ar. tiene la dirección del servidor web del Departamento de Computación.

```
www.dc.uba.ar.      IN      CNAME      www1.dc.uba.ar.
```

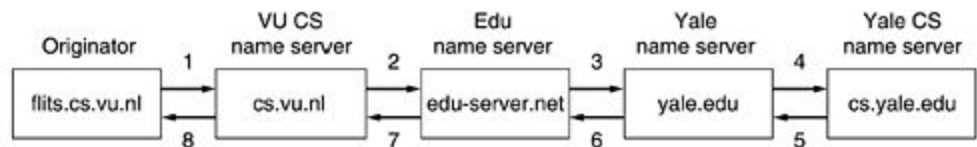
**www1.dc.uba.ar. IN A 157.92.27.127**

Le devuelve ésta dirección al DNS local, que a su vez se la devuelve al cliente.

- La siguiente figura ilustra el esquema de esta consulta (aunque para una consulta distinta):



- Esta forma de resolver una consulta se llama **iterativa**.
- Contrariamente, cuando cada servidor DNS se ocupa de resolver un dominio cuando no tiene la dirección en su tabla, se llama resolución **recursiva**.



- Lo bueno de la resolución iterativa, es que el DNS local podrá cachear las direcciones de todos los name servers intermedios. Entonces, cuando le llegue otra consulta a un dominio similar, podrá recurrir a un name server de una zona más próxima a la zona del dominio buscado, en lugar de tener que pasar por toda la secuencia de consultas desde root.
- **Glue records**
  - Muchas veces, los servidores DNS de una zona, tienen un nombre que está dentro de su misma zona. Por ejemplo, c.dns.ar. es name server de ar. y está ahí dentro. Si la respuesta del servidor DNS root fuera simplemente

**ar. IN NS c.dns.ar.**

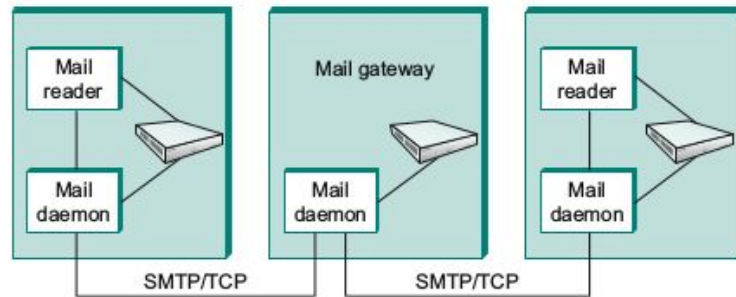
deberíamos ir a preguntar por la dirección de c.dns.ar. que está dentro de la zona de ar. Entonces, como aún no sabemos la dirección de c.dns.ar. (ni dns.ar. ni ar.) deberíamos ir a preguntarle a root nuevamente por esto. Y la respuesta sería la misma, generándose así un loop.

- Para solucionar esto es que se agrega un segundo registro  
**c.dns.ar. IN A 200.108.148.50**  
llamado glue record.

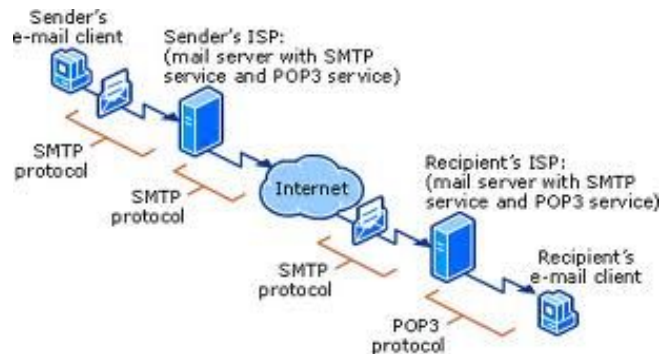
- **SMTP**

- Protocolo de transferencia de emails, entre servidores de mail.
- Un mail server es un servidor que contiene casillas de correo. El objetivo es enviar mails entre esas casillas.
- El protocolo es de tipo cliente-servidor. Corre sobre TCP. El servidor escucha el puerto 25.
- Los mensajes se interpretan como cadenas de caracteres ASCII. Tienen dos partes:
  - Header. Secuencia de líneas, cada una con la forma <tipo>: <valor>. Por ejemplo:  
From: John Doe <john@doe.com>  
To: [bob@gmail.com](mailto:bob@gmail.com)  
Subject: Hello
  - Body. El cuerpo del mensaje.
- **MIME (Multipurpose Internet Mail Extensions)**
  - Protocolo compañero de SMTP que extiende los tipos de información que se pueden mandar en un email: audio, video, imágenes, documentos PDF, etc.
  - Se añade al mensaje SMTP tradicional, líneas de éste protocolo, a fin de poder usar sus extensiones.
- Transferencia del mensaje:
  - El mail server en el que se encuentra la casilla de correo que quiere enviar el mail, se comunica con el mail server del correo del destinatario.
  - Utiliza TCP.
  - En general, la comunicación no se da entre el mail server emisor y destino, sino que el mensaje pasa por **mail gateways**. Éstos son servidores intermedios que forwarden mails a otros mail gateways o servers, hasta llegar al destino.

- Los mails gateways son necesarios porque, a veces, el mail server emisor no conoce exactamente el mail server donde está la casilla de correo del destinatario. Las casillas de un correo de un dominio (por ejemplo, gmail), están distribuidas sobre varios mail servers.



- Si la computadora donde se confecciona el email no es el mail server, la primera comunicación es entre la computadora del cliente y su mail server.



- El protocolo de transferencia es una secuencia de intercambio de mensajes. El emisor envía un comando (HELO, MAIL, DATA, etc.) junto con la información pertinente (por ejemplo, luego de DATA se coloca el contenido del mail), y el receptor emite una respuesta.

## ● POP3

- Además de poder transferir emails entre mail servers, necesitamos descargar los emails de nuestra casilla desde el mail server.
- POP3 es un protocolo que cumple ese propósito. Es de tipo cliente-servidor. Corre sobre TCP. El servidor escucha el puerto 110.
- Está centrado en el cliente:
  - Los correos se descargan del servidor (y son eliminados de allí) y se administran localmente.

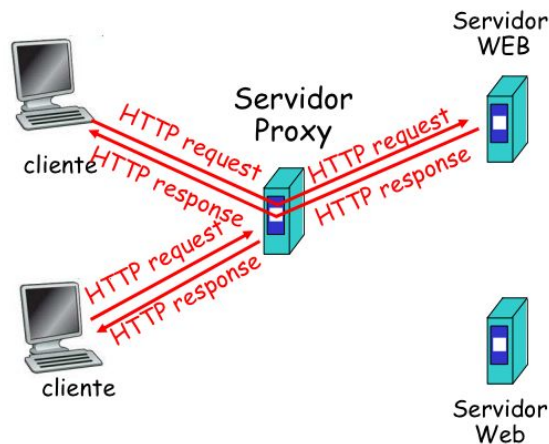
- Como toda la información se descarga, no hace falta conexión para acceder a los correos.
  - El backup pasa a ser responsabilidad del cliente.
  - El servidor solo tiene una *bandeja de entrada* de la que el cliente descarga los mensajes nuevos. La administración de mails (por ejemplo, la separación en carpetas) es responsabilidad del cliente.
- Cuando el usuario se comunica con su servidor, se realiza una etapa de autenticación.
- **IMAP**
  - Mismo propósito que POP3.
  - Tipo cliente-servidor. Corre sobre TCP. El servidor escucha el puerto 143.
  - Está centrado en el servidor:
    - Todos los mensajes quedan almacenados en el servidor. Cuando se desea acceder a la casilla de correo, debe descargarse la información desde el servidor.
    - Es útil cuando se accede a la casilla desde distintos hosts.
  - Tiene autenticación, al igual que POP3.
- **HTTP (HyperText Transfer Protocol)**
  - Protocolo para la descarga de documentos HTML (HyperText Markup Language) desde servidores web. Se utiliza para descargar páginas web, y así poder navegar en la WWW.
  - Es de tipo cliente-servidor. Corre sobre TCP. El servidor escucha el puerto 80. El browser (visualizador de páginas web) se comunica con el web server.
  - Todos los intercambios en HTTP son de tipo request/response, i. e. por cada solicitud hay una respuesta, y no se mantiene estado.
  - Request HTTP:
    - Se indica el tipo de operación a realizar.

Operation	Description
OPTIONS	Request information about available options
GET	Retrieve document identified in URL
HEAD	Retrieve metainformation about document identified in URL
POST	Give information (e.g., annotation) to server
PUT	Store document under specified URL
DELETE	Delete specified URL
TRACE	Loopback request message
CONNECT	For use by proxies

- Luego se incluyen líneas de encabezado.
- Los objetos requeridos (por ejemplo, un documento de un servidor web) se referencian usando **URL (Uniform Resource Locator)**. Por ejemplo, como <http://www.dc.uba.ar/materias/tc>. La característica de estas descripciones, es que indican *cómo ubicar a ese objeto*, dado que se incluye el nombre del web server, y una secuencia de directorios que desembocan en el archivo.
- Ejemplo:  
**GET <http://www.dc.uba.ar/materias/tc>**  
**HTTP/1.0**
- Response HTTP:
  - Se responde con un código, que indica el tipo de resultado (satisfactorio, error, etc.).
  - Luego vienen líneas de encabezado.
  - Luego los datos que acompañan a la respuesta.
- Conexión TCP
  - La versión original de HTTP (1.0) establecía una nueva conexión TCP para cada solicitud, aún si la comunicación era entre el mismo par de hosts. Como cada objeto referenciado en una página web (texto, imagen, etc.) hay que pedirlo, potencialmente por separado, la descarga de una página web simple resultaba en la apertura y cierre de múltiples conexiones TCP. Esto es evidentemente un desperdicio de recursos.
  - HTTP 1.1 introduce **conexiones persistentes**.
    - Permite al cliente y servidor, intercambiar múltiples mensajes sobre la misma conexión TCP.
    - Ventajas:
      - Reduce el overhead de la conexión a causa de TCP.
      - La conexión es efectivamente más rápida, ya que no hay que pasar por la etapa de slow start de TCP cada vez que se quiere enviar un mensaje.
    - Desventaja:
      - No es está claro cuándo debe cerrarse una conexión TCP. Esto es un problema para los servidores, que tienen que mantener muchas conexiones abiertas al mismo tiempo. Para esto se suele implementar un timeout: cuando no hay solicitudes durante un tiempo dado, se cierra la conexión.
- Caching HTTP



- Un usuario puede configurar su browser para acceder a la Web vía un **caching host** o **proxy**. Algunas veces, el ISP tiene un proxy por los que automáticamente pasan los pedidos HTTP de sus clientes, así que todo es transparente para ellos.
- Al proxy le llega un request HTTP de una URL. Si el objeto está en cache, retorna el objeto.
- Si no está en cache, hace un request HTTP del objeto al web server. Al recibirlo, lo guarda en cache y responde la request original al cliente.

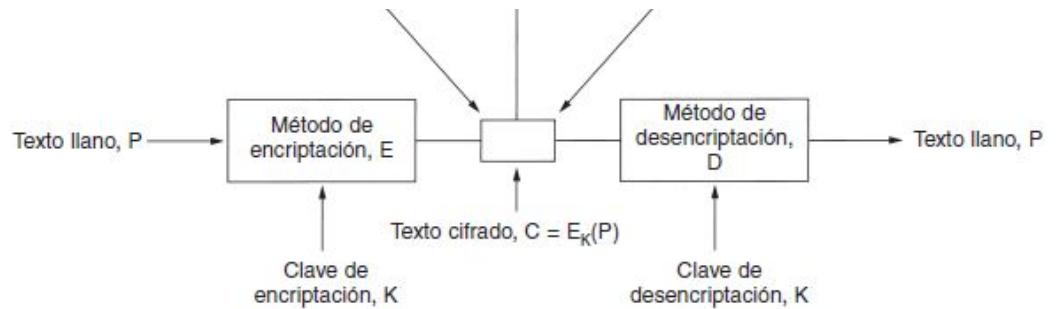


- Para determinar el tiempo de validez de un objeto, el web server incluye información de la expiración en su respuesta HTTP.
- Si el proxy contiene una entrada que ha expirado, puede hacerle una solicitud GET condicional (se incluye una línea específica en el header para que sea condicional) que sólo devuelve el objeto si fue modificado desde cierta fecha.

## Seguridad

- En una comunicación entre dos partes, solemos estar interesados en asegurar ciertas cosas:
  - **Autenticidad.** Cada parte es quien dice ser. Por ejemplo, si le solicito al web server de Google que me devuelva el contenido de [www.google.com](http://www.google.com), no quiero que alguien asuma la identidad de Google y me devuelva otra cosa.
  - **Integridad.** Un mensaje enviado no puede ser modificado durante su transmisión. No quiero que nadie modifique el contenido de [www.google.com](http://www.google.com) mientras viaja por Internet hasta mi computadora.

- **Confidencialidad.** Los mensajes sólo pueden ser interpretados por las dos partes de la comunicación. No quiero que nadie sepa que el contenido de Google que estoy solicitando.
- **No Repudiación.** Ninguna de las partes puede negar haber participado en una transacción.
- **Autorización.** La comunicación sólo puede ocurrir si el usuario (uno de los dos) está autorizado (tiene permisos).
- Se puede proveer seguridad en las distintas capas.
  - Nivel físico
    - Si el medio es guiado (un cable), se puede encerrar la línea en un tubo sellado conteniendo gas argón a presión. Si se pincha el cable, sensores de presión hacen saltar una alarma.
    - Para medios no guiados (wireless) se usan diversos algoritmos de cifrado (WEP, WPA, WPA2).
  - Nivel de enlace
    - Es posible encriptar el contenido de los frames de nivel de enlace, entre el emisor y receptor que comparten el enlace. El problema de esto es que vuelve más lento el ruteo, ya que cada router debe desencriptar el contenido recibido y encriptarlo para el siguiente router. Por otro lado, esta técnica es vulnerable a ataques dentro de cualquiera de los routers del camino.
  - Nivel de red
    - Se puede utilizar el protocolo IPSEC.
- Criptografía
  - **Cifrado**
    - Transformación carácter por carácter o bit por bit, sin importar la estructura lingüística del mensaje.
  - **Código**
    - Reemplaza una palabra con otra palabra o símbolo. Le importa la estructura del mensaje.
  - **Principio de Kerckhoff.** Los algoritmos de cifrado deben ser públicos; sólo las claves deben ser secretas.
    - Las claves son los parámetros de los algoritmos de cifrado.
    - Este principio se basa en que cambiar la clave es mucho más fácil que cambiar un algoritmo.
    - Cuanto más grande es la clave, más tiempo suele ser necesario para romper un cifrado (descifrar sin conocer la clave). Para empezar, hacer fuerza bruta sobre todas las posibles claves de un largo fijo, toma tiempo exponencial en el largo.



- Métodos de cifrado clásicos
  - **Cifrado por Sustitución.** Cada letra o grupo de letras es reemplazado por otra. El orden se preserva.
    - **Cifrado César.** Cada letra se mapea con otra, que se encuentra un número fijo de posiciones más adelante en el alfabeto.
      - Se puede descifrar más o menos rápido.
      - Una forma es, asumiendo que se sabe que se utilizó un Cifrado César, se prueban los 26 posibles desplazamientos.
      - Otra forma es haciendo un **análisis de frecuencia**. Por ejemplo, como las letras más frecuentes del castellano son A y E, podemos intentar descifrar asumiendo que las letras que más aparezcan en el mensaje cifrado corresponden a alguna de esas dos.
  - **Cifrado por Transposición.** Reordena las letras, pero no las cambia.
    - **Transposición columnar.** Ver [https://en.wikipedia.org/wiki/Transposition\\_cipher#Columnar\\_transposition](https://en.wikipedia.org/wiki/Transposition_cipher#Columnar_transposition) que está muy bien explicado.
  - **One-time Pads (rellenos de una sola vez).** Elegir un gran random bit string como clave, de la misma longitud que el mensaje a cifrar. Hacer XOR entre la clave y el mensaje.
    - Shannon demostró que es inviolable. Concretamente demostró que el texto cifrado no proporciona absolutamente ninguna información sobre el mensaje original.
    - Problema: cómo distribuir y proteger la clave. Además, cuando es utilizada más de una vez, pierde la propiedad de ser indescifrable.

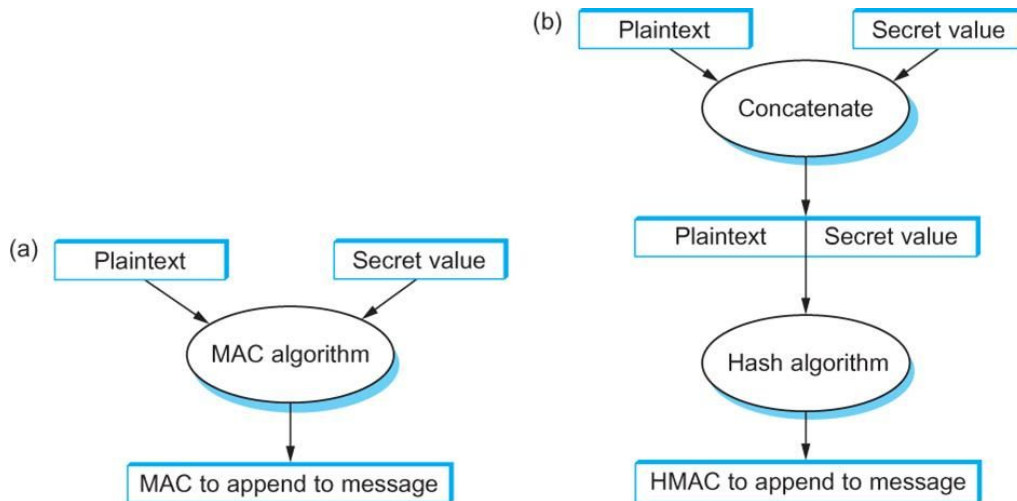
- **Cifrado por bloques.** Se toma el mensaje de a bloques de  $n$  bits, y se transforma cada bloque utilizando la clave.
- **Cifrado simétrico**
  - Ambos participantes de la comunicación comparten la misma clave.
  - La misma clave se usa para encriptar y desencriptar. Por lo tanto, la clave sólo debe ser conocida por los participantes.
  - Los más usados:
    - **DES (Data Encryption Standard)**
      - Cifrado por bloques. Orientado a bits.
      - Las claves son de 64 bits (es decir, va cifrando de a bloques de 64 bits).
      - Usa tanto transposición como sustitución.
      - Cada bloque ejecuta 16 iteraciones con distintas claves (obtenidas a partir de una única clave inicial).
      - Si bien no se conocen ataques más rápidos que fuerza bruta, hoy en día no toma demasiado tiempo hacer fuerza bruta sobre claves de 64 bits.
    - **3DES**
      - Aplicación sucesiva de DES, para hacer más difíciles los ataques de fuerza bruta.
      - Se ejecutan 3 pasadas de DES:
        - Se encripta usando una clave  $K_1$ .
        - Se desencripta usando una clave  $K_2$ .
        - Se encripta usando una clave  $K_3$ .
      - Cuando  $K_1 = K_2$ , el algoritmo es DES.
    - **AES (Advanced Encryption Standard)**
      - Es el recomendado hoy en día. Admite claves mucho más largas (128, 192 ó 256 bits).
- **Cifrado asimétrico (o de clave pública)**
  - Cada participante tiene dos claves, una pública y una privada.
  - La clave pública puede ser conocida por cualquiera. La privada sólo es conocida por el dueño.
  - Si  $P_U$  es la clave pública y  $P_R$  es la privada, entonces  $M = D_{P_R}(E_{P_U}(M))$ . En otras palabras, si encriptamos con la clave pública, podemos desencriptar con la privada.
  - También podemos usar las claves en el otro sentido:  $M = D_{P_U}(E_{P_R}(M))$ .
  - Si Bob le quiere mandar un mensaje a Alice, encripta el mensaje usando la clave pública de Alice, y se lo manda. Nadie puede leer el

mensaje, porque sólo se puede descryptar con la clave privada, que sólo la tiene Alice.

- Observar que un par de claves pública + privada sólo pertenece a uno de los participantes, y sólo permite la comunicación en un sentido. Por lo tanto, en una comunicación bidireccional, son necesarios dos pares.
- Los más usados:
  - **Diffie-Hellman**
  - **RSA**
    - Su seguridad se basa en que factorizar es un problema difícil (no se conoce algoritmo polinomial).
    - Requiere claves de, al menos, 1024 bits, mucho más grandes que los algoritmos de cifrado simétrico. Esto se debe a que es mucho más rápido romper RSA factorizando un número entero que hacer fuerza bruta sobre el espacio de claves candidatas. (Igualmente, factorizar un número es exponencial.)
  - **ElGamal**
    - Su seguridad se basa en que el problema del logaritmo discreto es difícil.
- Son más lentos que los cifrados simétricos. Es por esto que sólo se los usa al principio de una comunicación segura, para autenticación y para establecer una clave de sesión para usar un cifrado simétrico (todo esto se explica a continuación).
- **Autenticación**
  - La simple encriptación de los mensajes provee confidencialidad, pero no permite asegurar integridad ni autenticidad.
  - Para asegurar integridad y autenticidad simultáneamente, se utiliza un **autenticador**. Un autenticador es un valor (texto cifrado) que se incluye en el mensaje transmitido.
  - Para asegurar integridad, se utiliza un **checksum criptográfico** o **hash criptográfico**. Esto es una función de hash, diseñada para exponer modificaciones del mensaje por un adversario (una persona que quiere intervenir en la comunicación).
  - El checksum criptográfico se aplica al mensaje que se quiere enviar. El output se denomina **message digest**, y se appendea al mensaje a enviar.
  - El receptor recibe el mensaje junto con el digest, hasha el mensaje nuevamente y lo compara contra el digest.

- La característica de este digest es que ante una mínima modificación al mensaje, su digest cambia completamente. Notar que, sólo con este digest no podemos garantizar autenticidad, de modo que todo ésto sería en vano.
- Autenticador 1: **digest encriptado**
  - Para efectivamente garantizar autenticidad además de integridad, se encripta el digest con una clave (la clave simétrica si el cifrado es simétrico, o la propia clave privada si es asimétrico), de modo tal que un adversario podría interceptar el mensaje, cambiarlo por otro y hashearlo, pero no podrá encriptarlo.
  - Si se encripta con cifrado simétrico, el receptor desencryptará con la misma clave. Si se encripta con cifrado asimétrico, se encripta con clave privada y desencrypta con pública.
- Propiedades deseables de un hash criptográfico h:
  - Resistencia a preimagen
    - Dado un digest x, debe ser difícil encontrar un mensaje m tal que  $h(m) = x$ .
    - Ataque de preimagen: se conoce el digest x, y a partir de éste se deduce el mensaje original.
  - Resistencia a colisiones
    - Dado un mensaje m, es difícil encontrar un mensaje m' tal que  $h(m) = h(m')$
    - Ataque de colisiones: cambiar el mensaje m por m', manteniendo el digest (que podría estar encriptado con una clave que no conocemos).
- Hash criptográficos más usados:
  - MD5 (Message Digest 5)
    - Digest de 128 bits.
  - SHA-1 (Secure Hash Algorithm 1)
    - Digest de 160 bits.
- Autenticador 2: **MAC (Message Authentication Code)**
  - Otra forma de generar un autenticador.
  - Utiliza una función de hash parametrizada con un valor secreto (un valor sólo conocido por el emisor y el receptor).
  - Aplicamos la función de hash con clave al mensaje. El resultado se denomina MAC y se appendea al resto del mensaje.
  - **HMAC (Hashed Message Authentication Code)**
    - Variante de MAC, que no usa una función de hash con clave.

- Concatenamos el mensaje y el valor secreto, y hasheadmos con cualquier hash criptográfico. El resultado se denomina HMAC y se appendea al resto del mensaje.



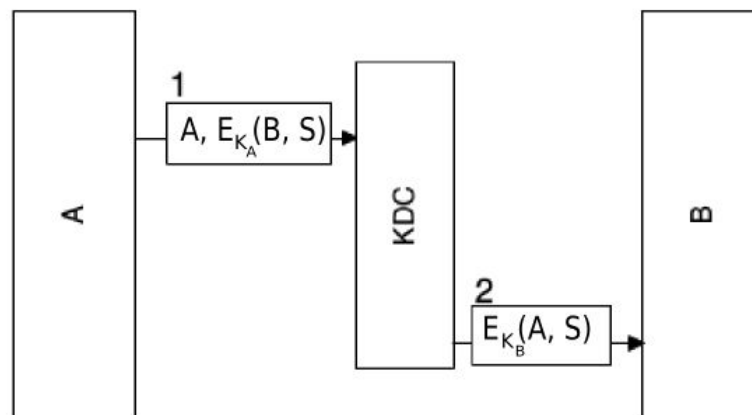
- **Firma digital**

- Es un mecanismo que garantiza autenticidad, integridad y no repudiación.
- Recordemos que un digest encriptado aseguraba autenticidad e integridad.
- Si el digest es encriptado utilizando una clave privada, entonces también garantiza no repudiación, pues no hay otra entidad que pueda encriptar con esa clave. El receptor puede usar la clave pública para desencriptar y corroborar el digest.
- Un método utilizado para firma digital es SHA-1 + RSA.
- La firma digital (y los digest encriptados, más generalmente) aseguran autenticidad en el sentido de que sabemos que el emisor del mensaje es el dueño de la clave privada. Ahora bien, ¿cómo sabemos si el dueño de la clave privada, asociada a la pública que estamos usando, es efectivamente la persona que creemos que es? En otras palabras, si nos llega una clave pública de alguien que dice ser Google, ¿cómo sabemos que realmente es Google?

- **Predistribución de claves públicas**

- Necesitamos una forma de ligar una clave pública a una identidad.
- Idea: si tenemos garantía de la identidad de una entidad X (es decir, tenemos su clave pública y confiamos que es de X), y X firma digitalmente un documento que asocia a la entidad Y con una clave pública, entonces podemos confiar en la identidad de Y.
- Un documento firmado digitalmente que asocia una identidad con una clave pública (el documento debe contener tanto el nombre de la entidad como la clave pública) se llama **certificado de clave pública**.

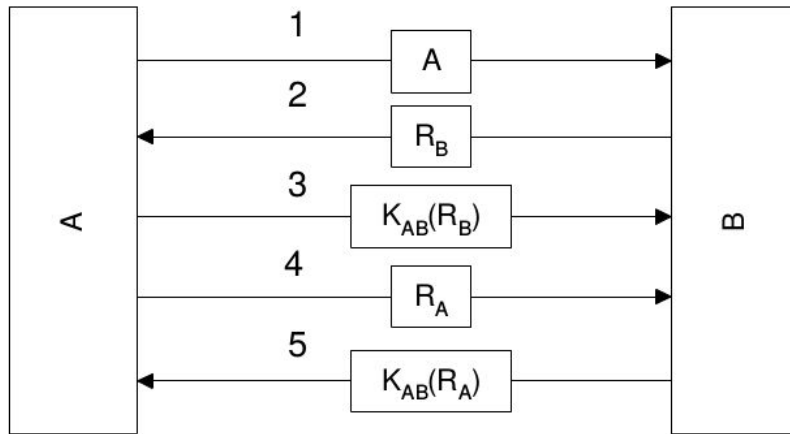
- Como ahora confiamos en Y, Y puede certificar otras entidades. Esto da lugar a una **chain of trust**.
- Las entidades base, en las que confiamos a ciegas (y de las que tenemos sus claves públicas), que son la raíz de las chain of trust se llaman **Autoridades Certificantes (CA)**.
- Cuando Google quiere distribuir su clave pública, le solicita a una CA que firme digitalmente un certificado de clave pública suyo. Luego simplemente distribuye ese certificado firmado. Un usuario que confía en la CA, podrá corroborar que la clave pública de Google es la que afirma el certificado.
- **Predistribución de claves simétricas**
  - Es un problema más complicado que la predistribución de claves públicas, porque ahora necesitamos una clave única para cada par de participantes que quieran comunicarse (si más de dos personas conocen una clave, entonces hay un tercero que puede leer los mensajes de la comunicación).
  - Si tengo N participantes, necesito  $O(N^2)$  claves.
  - Se puede bajar a  $O(N)$  claves si hay una entidad central que comparte un secreto con cada uno de los N participantes.
  - La entidad central se denomina **Key Distribution Center (KDC)**
    - El KDC comparte una clave con cada uno de los participantes. Para una entidad X, llamemos  $K_X$  a la clave compartida entre X y el KDC.
    - Cuando A se quiere comunicar con B, se lo informa al KDC. Le envía al KDC una clave de sesión S (la clave simétrica para la comunicación entre A y B), encriptada con  $K_A$ .
    - El KDC desencripta, y encripta nuevamente S, ahora con  $K_B$ , y se la manda a B.



- A y B pueden comenzar a hablar usando S.
- **Timeliness** (temporalidad)

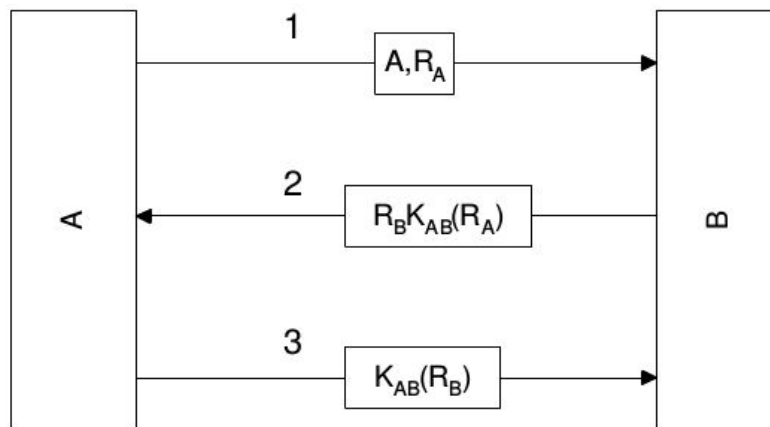


- Hay un factor más, que no hemos considerado aún, que hace a la autenticidad.
- Supongamos que estamos comprando un producto en Amazon.com. En el momento en que le damos el OK a Amazon para comprar un producto, un adversario lee la comunicación. Como todo está encriptado, no puede saber lo que está pasando. Sin embargo, comienza a replicar el mensaje nuestro, enviándoselo múltiples veces a Amazon. Amazon no debería procesar la compra múltiples veces.
- Este ejemplo, medio tonto, muestra que hay un problema. Los mensajes que recibe Amazon son auténticos: los escribimos nosotros y nadie los alteró en el camino.
- La razón por la que no son auténticos es que *no llegaron en tiempo*. El primero llegó en tiempo, pero las réplicas sucesivas no. Por ende, para asegurar autenticidad, debemos asegurar temporalidad.
- **Replay Attack**
  - Ataque similar al descrito, en el que el adversario transmite una copia de un mensaje enviado legítimamente.
  - **Supress-replay Attack**. En lugar de repetir el mensaje, se lo intercepta, demora y transfiere posteriormente.
- **Protocolos de autenticación**
  - Ya vimos que con autenticadores podemos asegurar que un mensaje fue creado por el dueño de una clave pública, y que no ha sido alterado durante la transmisión. Queremos sumar el tema de la temporalidad. Asumimos que ya confiamos en la identidad del dueño de una clave pública (o no, pero no nos importa).
  - Queremos protocolos, que cumplan todo lo anterior, para comenzar una comunicación entre dos partes.
  - **Autenticación con clave simétrica compartida**
    - Llamemos  $K_{AB}$  a la clave que comparten A y B.
    - A envía su identidad a B (indicando que quiere comunicarse).
    - B no sabe si ese mensaje es realmente de A. Elige un **challenge**  $R_B$  (un número al azar grande) y se lo manda a A sin encriptar.
    - A encripta  $R_B$  con  $K_{AB}$ , y se lo manda a B.
    - Se repite el procedimiento en el otro sentido. A le manda un challenge  $R_A$  a B, y B le responde encriptándolo con  $K_{AB}$ .

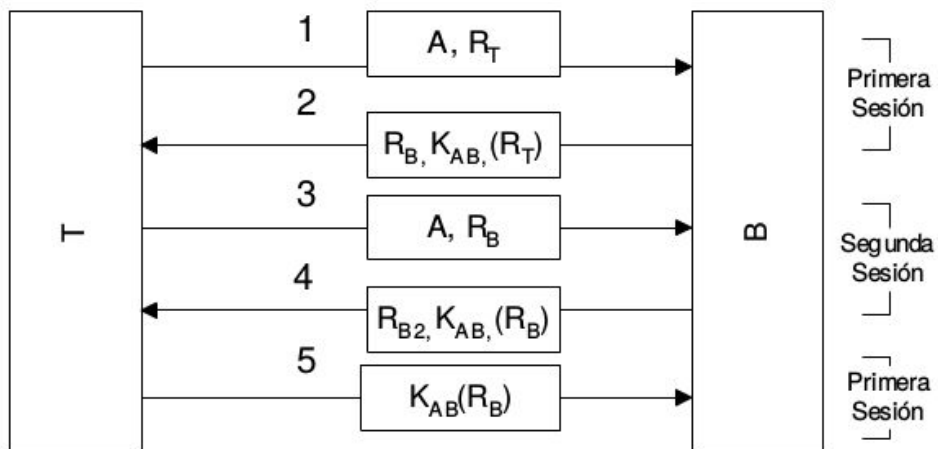


○ **Autenticación con clave simétrica compartida simplificado**

- Similar al anterior pero nos ahorramos un par de pasitos.



- Este esquema, si bien es más corto, es propenso a un **ataque por sesiones**



- Lo que sucede aquí es que T (un adversario) se quiere hacer pasar por A. Entonces le envía a B un challenge  $R_T$  (cualquiera, no importa) pero le dice que es A.
  - B le responde el challenge  $R_T$  encriptado (a T no le importa) y el challenge  $R_B$ .
  - T no sabe encriptar  $R_B$  porque no tiene la clave  $K_{AB}$ . Entonces, indirectamente hace que B lo encripte. ¿Cómo? Inicia otra sesión con B, y le manda como challenge a  $R_B$ .
  - Reglas de diseño para evitar este tipo de ataques:
    - El que inicia la transmisión debe probar su identidad antes que el receptor.
    - Usar claves diferentes para la verificación de identidades. Es decir, A debería usar una clave  $K_{AB}$  y B una clave  $K'_{AB}$ , ambas compartidas.
    - Elegir los challenges de conjuntos diferentes. Por ejemplo, el que inicia la comunicación elige un challenge par, el receptor un impar.
- Ataques de red
  - **Sniffing**
    - Escuchar los datos de la red sin interferir la conexión.
    - Protección: encriptar los datos que viajan por la red.
  - **Spoofing**
    - Hacerse pasar por otro, interviniendo la conexión.
  - **Hijacking**
    - Robar una conexión después que una parte legítimamente se autenticó ante otra.
  - Ingeniería social
    - Phishing
  - Explotar bugs de software
  - Caballo de Troya
  - DoS